



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Introduction

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Security

- Making a system (a software, a computer, a network, etc.) secure, requires three types of security:

- **physical security**

- physically limit the access to the system
 - servers behind a locked door and only a privileged set of employees have access to it
 - use of cameras, card readers, and biometric locks
 - protection against information leakage
 - e.g. by shredding documents before they're thrown away

- **technological security**

- communication and data security
 - software security
 - application security
 - OS security

- **good policies and practices**



Security Service

- Something that enhances the security of the systems and the information transfer
 - **aims to protect data, systems, user information**
 - **intended to counter security attacks**
- A processing or communication service that is provided by a system to give a specific kind of protection to system resources
 - **RFC 4949 , "Internet Security Glossary"**
 - <https://tools.ietf.org/html/rfc4949>
- Makes use of one or more security mechanisms to provide the service
- Replicates functions normally associated with physical objects/documents
 - **e.g. signatures, dates, proof of reception, notarization, recording, etc.**



Security Services (cont.)

- Some security services:

- Confidentiality
- Data integrity and message authentication (authenticity)
- Peer entity authentication (identification)
- Authorization and access control
- System integrity and availability
- Accountability and non-repudiation
- Anonymity



Security Services (cont.)

● Confidentiality

- **protects data against unauthorized disclosure**
 - It is the property that information is not made available to unauthorized entities
 - related to
 - data
 - » data confidentiality
 - entities involved in the communication
 - » anonymity



Security Services (cont.)

- Data integrity and message authentication

- **data integrity**

- the property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner
 - protects against unauthorized changes to data by ensuring that changes to data are detectable
 - in general it can only detect a change

- **data origin authentication**

- provides for the corroboration of the source of a data unit
 - this service verifies the identity of a system entity that is claimed to be the original source of received data
 - usually provided together with data integrity → message authentication

- **message authentication (authenticity)**

- both data integrity and origin authentication
 - in general, authenticity implies integrity but integrity doesn't imply authenticity



Security Services (cont.)

- Identification (peer entity authentication)
 - **verification of the identity of a peer entity**
 - before the establishment of a communication or the access to a resource/service
- Authorization and access control
 - **authorization**
 - verification of the permission to access a resource or system
 - manage access rights/privileges
 - **access control**
 - ability to limit and control the access to a systems
 - protection of system resources against unauthorized access



Security Services (cont.)

- System integrity and availability

- **system integrity**

- the quality that a system has when it can perform its intended function
 - protects system resources against unauthorized change, loss, or destruction

- **availability**

- protects a system to ensure its availability
 - addresses the security concerns raised by denial-of-service (DoS) attacks



Security Services (cont.)

● Accountability and non-repudiation

➤ **accountability**

- property of a system or system resource that ensures that the actions of an entity may be traced uniquely to that entity

➤ **audit**

- service that records information needed to establish accountability

➤ **non-repudiation**

- provides protection against false denial of an action
 - it provides evidence that can be stored and later presented to a third party
- in case of a communication, it prevents either sender or receiver from denying a transmitted message
 - the receiver can prove that the sender in fact sent the message
 - » non-repudiation with proof of origin
 - the sender can prove that the receiver in fact received the message
 - » non-repudiation with proof of receipt



Security Services (cont.)

● Anonymity

➤ **The condition of an identity being unknown or concealed**

- An application may want to maintain anonymity of users or other system entities, perhaps to preserve their privacy
- When two (or more) parties interact without letting the possible observers detect such relation
 - who is talking with whom



Security Mechanisms

- Security services are provided by means of different security functions/ mechanisms
 - **they can be included in appropriate communication layer**
- Examples of security mechanisms are:
 - **enciphering**
 - **authentication exchange**
 - **data integrity check**
 - **digital signature**
 - **notarization (third-party authentication)**
 - **access control**
 - **traffic padding**
 - **routing control**
 - **etc.**

Relationship Between Security Services and Mechanisms

Service	Mechanism							
	Encipherment	Digital signature	Access control	Data integrity	Authentication exchange	Traffic padding	Routing control	Notarization
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Non-repudiation		Y		Y				Y
Availability				Y	Y			



Classification of Security Attacks

- **Passive attacks** (eavesdropping on, or monitoring of transmissions):
 - **Interception (snooping)**
 - obtain message contents (attacks confidentiality)
 - **Traffic analysis**
 - monitor traffic flows (attacks confidentiality)
- **Active attacks** (modification of data stream):
 - **Spoofing**
 - fabrication of messages with a fake source entity (attacks authenticity)
 - **Tampering**
 - modify of message content (insert, cancel, modify data) (attacks integrity)
 - **Replay/Reflection**
 - replay previous messages to/from the same or different entity (attacks authenticity)
 - **Repudiation**
 - deny having sent or received a message (attacks Non-reputation)
 - **Denial of Service (DOS)**
 - Interruption of a network or application service (attacks availability)



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Secret Key (symmetric) Cryptography



Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Cryptography

- Study of mathematical techniques related to information and communication security in the presence of third party adversaries
- The most widely used tool used by different security services
 - **not the only one**
- Can be used for:
 - **Confidentiality**
 - **Data integrity**
 - **Authentication**
 - **Non-repudiation**



Different cryptographic algorithms

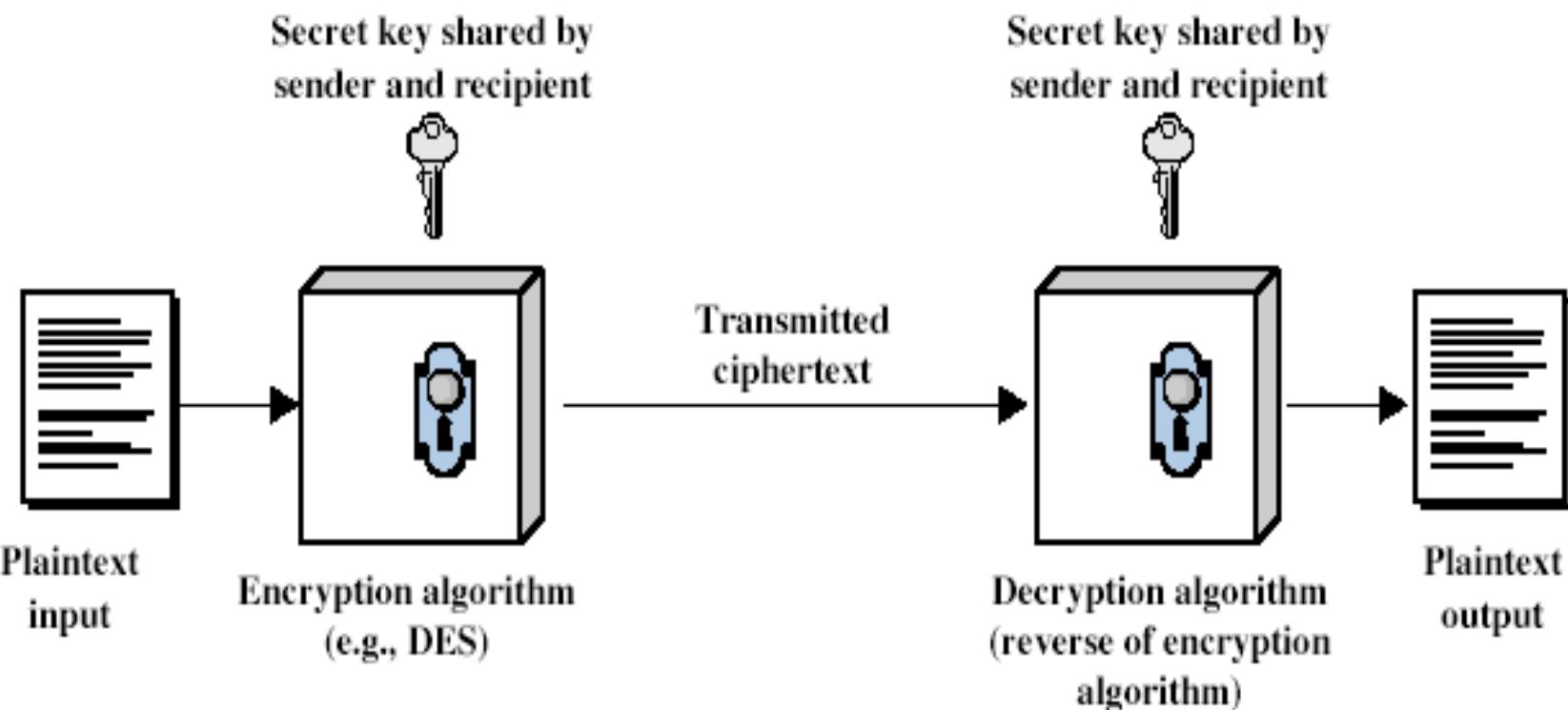
- Symmetric cryptography (Secret key cryptography)
 - **the two communication parties share a common secret (key)**
- Asymmetric cryptography (Public key cryptography, or private/public key cryptography)
 - **two different keys are used for two opposite functions (e.g. encryption and decryption)**
 - **one key can be publicly available (public key); the other is maintained secret for the owner (private key)**
- Hash algorithm (message digest/one way transformation)
 - **one-way transformation that maps a variable length message to a fixed length bit string**
 - **a variant is a MAC function**
 - includes a key



Symmetric Cryptography

- Or conventional / secret-key / single-key
 - **sender and recipient share a common key**
- All classical encryption algorithms are secret-key
 - **was the only type prior to invention of public-key in 1970's**
- Generally used for protecting (through encryption) some data stored in a repository or sent to a remote entity
- Designed to take a reasonable-length key (e.g. 128 bits) and generating a one-to-one mapping from cleartext to ciphertext that “looks like completely random”, to someone doesn't know the key

Symmetric Cipher Model





Symmetric Cipher Model (cont.)

- Plaintext - the original message (m)
- Ciphertext - the encoded message (c)
- Key - info used known only to sender/receiver (k)
- Cipher - algorithm for transforming plaintext to ciphertext
- Two functions:
 - **Encipher (Encryption) - converting plaintext to ciphertext**
 - $c = E(k, m) = E_k(m)$
 - can be either a deterministic or randomized function
 - **Decipher (Decryption) - recovering ciphertext from plaintext**
 - $m = D(k, c) = D_k(c) = D_k(E_k(m))$
 - deterministic
- Common symmetric algorithms:
 - DES, 3DES, RC4, IDEA, AES



Symmetric Cipher Model (cont.)

- In theory, the security of a cipher might rest in the secrecy of its restricted algorithm
 - $c = E(m)$, $m = D(c)$

however:

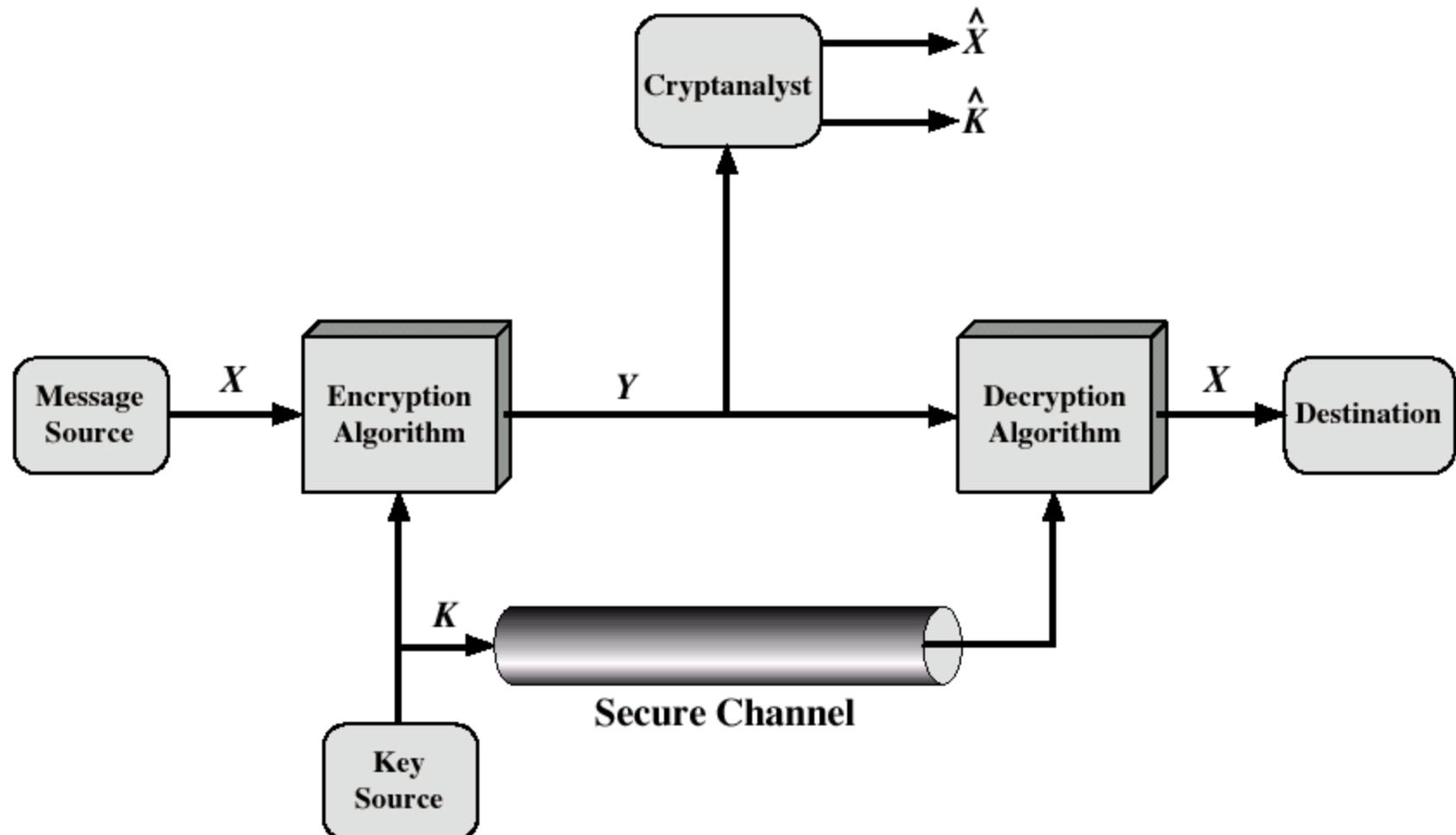
 - whenever a user leaves a group, the algorithm must change
 - could be scrutinized by people smarter than you
- In practice, the encryption algorithm is usually not secret
 - **keys are used and the security relies on the secrecy of keys**
 - selected from a large set (a keyspace), e.g., a 256-bit number $\rightarrow 2^{256} \approx 10^{77}$ values!
 - $c = E(k,m) = E_k(m)$, $m = D(k,c) = D_k(c)$
 - change of authorized participants requires only a change in key
 - the robustness of the algorithm is usually proportional to the key length
 - e.g. 40 bit (weak), 128 bit (strong)
 - **Kerckhoffs' principle: Security should be based on secrecy of the key, not the details of the algorithm**
 - Jean Guillaume Hubert Victor Francois Alexandre Auguste Kerckhoffs von Nieuwenhof, "La Criptographie Militaire", 1883



Symmetric Cipher Model (cont.)

- The two parties must know the algorithm to be used and must share a secret key
 - **requires an initial phase where the two parties exchange in secure manner the shared secret key**
 - implies a secure channel (or method) to distribute the key

Symmetric Cipher Model (cont.)





Threat model

- This specifies what “power” the attacker is assumed to have, without placing any restrictions on the adversary’s strategy
- Plausible options for the threat model are:
 - **Ciphertext-only attack**
 - **Known-plaintext attack**
 - **Chosen-plaintext attack**
 - **Chosen-ciphertext attack**



Ciphertext only - Attack

- The bad guy has seen (and presumably stored) some ciphertext that can be analyzed
 - **he/she should be able to recognize when he/she has succeeded (often called *recognizable plaintext* attack)**
 - for example in case of normal text or known document formats
 - **it is necessary to have enough ciphertext**
- It is the hardest attack to carry on
 - **the opponent has the least amount of information to work with**



Known plaintext - Attack

- The bad guy knows a <plaintext, ciphertext> pair
- From that pairs, the attacker can try to figure out the mapping of some fraction of the text
- How it is possible to obtain the plaintext?
 - **the secret data does not remain secret forever (e.g. the name of an attacked city)**
 - **or the opponent may have knowledge of what is in the message**
 - certain key words (e.g. in the header of the message)
 - expected patterns (e.g. PostScript, etc.)
 - probable-word attack
 - » between *Ciphertext only* attack and *Known plaintext* attack
- Some cryptographic schemes might be good enough to be secure against *ciphertext only* attacks but not against to *known plaintext* attacks
 - **in these cases, it is important to minimize the possibility for a bad guy to obtain <m,c> pairs**



Chosen plaintext (or ciphertext) - Attack

- The opponent can choose any plaintext and get the corresponding ciphertext from the system (or the contrary)
 - e.g. there is a transmission service that encrypts and transmits messages; the bad guy can ask the transmission service to transmit any plaintext he/she wants
- Some cryptographic schemes might be good enough to be secure against *ciphertext only* attacks and *known plaintext* attacks but not against to *chosen plaintext* attacks



Cryptography Attacks

- There are some general approaches to attacking a conventional encryption scheme:

- **Cryptographic analysis (cryptoanalysis)**

- based on the type of the cryptographic algorithm, tries to exploit some characteristic of the algorithm and/or properties of some previous plaintext/ciphertext pairs to deduce a plaintext and/or key
 - does not require to obtain the encryption key for deduce the plaintext

- **Brute-force (search) attack**

- tries every possible encryption/decryption (e.g. by trying all possible keys)
 - it may require the visit of all key space
 - The average number of required attempts is the half of the number of possible keys
 - it requires to be able to recognize when the correct plaintext/ciphertext has been obtained

- **Side-channel attacks**

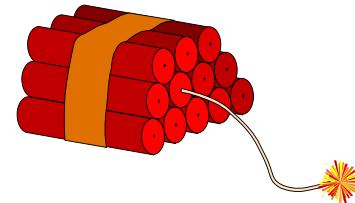
- use information from the physical implementation of a cryptosystem



Cryptoanalysis

- Based on the type of the cryptographic algorithm, tries to exploit some characteristic of the algorithm and/or properties of some previous plaintext/ciphertext pairs
 - **to deduce a plaintext and/or key**
 - **does not require to obtain the encryption key for deduce the plaintext**

Brute force attack



- Method of defeating a cryptographic scheme by trying a large number of possibilities
 - **for symmetric-key ciphers it typically means a brute-force search of the key space**
 - testing all possible keys in order to recover the plaintext used to produce a particular ciphertext
- In most schemes, the theoretical possibility of a brute force attack is recognized, but it is set up in such a way that it would be computationally infeasible to carry out
- The attacker has to determine when he succeeded
 - **By obfuscating the data to be encoded, brute force attacks are made less effective as it is more difficult to determine when one has succeeded in breaking the code**



Brute Force Search - Example

Key Size (bits)	Number of Alternative Keys	Time Required at 1 Decryption/ μ s	Time Required at 10^6 Decryptions/ μ s
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\mu\text{s} = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\mu\text{s} = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\mu\text{s} = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\mu\text{s} = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\mu\text{s} = 6.4 \times 10^{12}$ years	6.4×10^6 years



Side channel attack

- Any attack based on information gained from the physical implementation of a cryptosystem, rather than theoretical weaknesses in the algorithms
 - e.g. timing information, power consumption
- General classes of side channel attack include:
 - Timing attack — attacks based on measuring how much time various computations take to perform
 - Power monitoring attack — attacks which make use of varying power consumption by the hardware during computation
 - TEMPEST (aka Van Eck or radiation monitoring) attack — attacks based on leaked electromagnetic radiation which can directly provide plaintexts and other information



Side channel attack (cont.)

- In all cases, that physical effects caused by the operation of a cryptosystem can provide useful extra information about secrets in the system
 - **about the cryptographic key, partial state information, full or partial plaintexts and so forth**
- Side-channel attacks require considerable technical knowledge of the internal operation of the system on which the cryptography is implemented



Cryptographic break

- A cryptographic "break" is anything faster than an exhaustive search (brute force attack)
 - **example, an attack against a 128-bit-key cipher requiring “only” 2^{120} operations (compared to 2^{128} possible keys) would be considered a break even though it would be, at present, quite infeasible**
- The loss of a key (also without cryptoanalysis or brute-force attack) is called a “compromise”



Computational and Unconditional Security

● Unconditional security

- **an encryption scheme is unconditionally secure if no matter how much computer power is available, the cipher cannot be broken**
 - the ciphertext provides insufficient information to determine the corresponding plaintext
 - e.g. OTP (One Time Pad) chiper

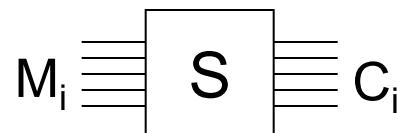
● Computational security

- **however cryptographic algorithms are often not impossible to attack**
 - e.g. brute force attack
- **an encryption scheme is computationally secure if given limited computing resources the cipher cannot be broken**
 - e.g. the time required to break the cipher exceeds the useful lifetime of the information
 - depends on the attack complexity and cost
 - processing complexity: a large number of operations required (long time)
 - data complexity: a large number of expected inputs (e.g., ciphertext)
 - storage complexity: a large amount of storage units required
 - requires the estimation of computation power of the opponent

Classical encryption techniques

Substitution Ciphers

- Substitution is a classical encryption technique
- Letters of plaintext are replaced by other letters or by numbers or symbols
- If plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns



M	C
0000	1101
0001	1001
0010	0111
0011	1000
:	:
1111	0011

Substitution Table

- Simple examples of classical substitution ciphers
 - monoalphabetic substitution with shift (e.g. Caesar cipher)
 - monoalphabetic substitution (monoalphabetic cipher)
 - polialphabetic substitution (polialphabetic cipher)



Caesar Cipher

- Earliest known substitution cipher
 - by Julius Caesar
 - first attested use in military affairs
 - it is a monoalphabetic substitution with shift
- Replaces each letter by 3rd letter on
- Example:

meet me after the toga party
PHHW PH DIWHU WKH WRJD SDUWB



Caesar Cipher (cont.)

- Can define transformation (substitution) as:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- Mathematically give each letter a number

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

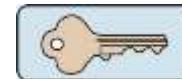
- Then have Caesar cipher as:

$$C = E(P) = (P + k) \bmod 26, \text{ with } k=3$$

$$P = D(C) = (C - k) \bmod 26, \text{ with } k=3$$

- If k is generic (and secret), we have a *Shift cipher*

➤ **k is the key, with $K \in \{0, 1, \dots, 25\}$**





Cryptanalysis of a Shift Cipher

- Only have 26 possible ciphers
 - A maps to A,B,..Z
 - If the mapping of one letter is discovered, the entire transformation is found
- Given ciphertext, could just try all shifts of letters
 - brute force search
 - e.g. break ciphertext "GCUA VQ DTGCM"
- Do need to recognize when have plaintext

KEY	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	kccr kc ydrcc rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkbdi
20	vnnn vn jocna cqñ cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc



Monoalphabetic Substitution Ciphers

- Rather than just shifting the alphabet
- Could shuffle (permute) the letters arbitrarily
- Each plaintext letter maps to a different random ciphertext letter
- Example:

➤ **Substitution table:**

abcdefghijklmnopqrstuvwxyz
DKVQFIBJWPESCXHTMYAUOLRGZN

- **plaintext:** if we wish to replace letters
➤ **ciphertext:** WIRFRWAJUHYFTSDVFSUUUFY

- Note: the secret substitution can be seen as the secret key
- Now have a total of $26! \approx 4 \times 10^{26}$ keys
 - with so many keys, might think is secure
 - but would be wrong!
 - cryptoanalysis based on text frequency and correlation



Cryptoanalysis of Monoalphabetic Cipher

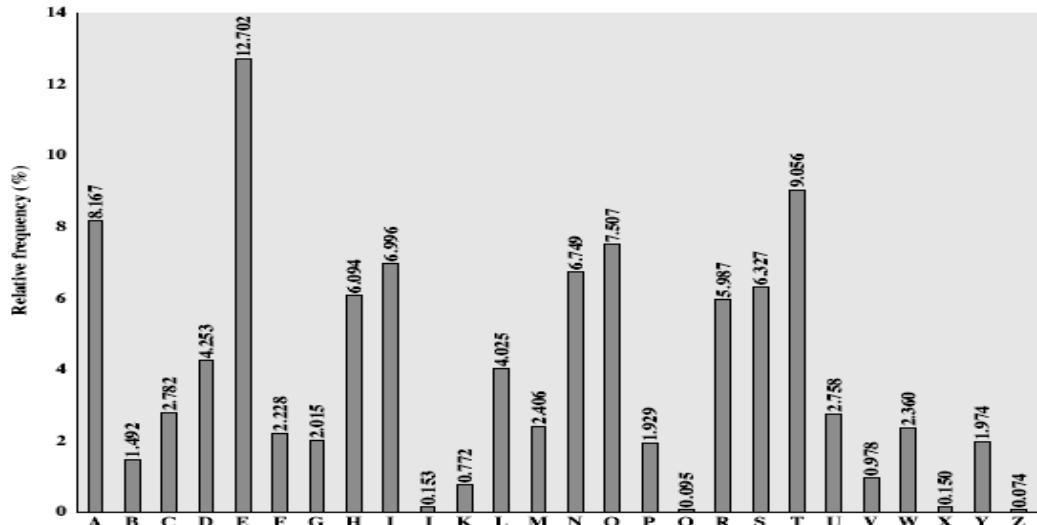
- Main problem with monoalphabetic substitutions is text redundancy
 - **non uniform frequency distributions and correlation**
- In case of human languages:
 - **letter frequencies**
 - in English 'e' is by far the most common letter, then T,R,N,I,O,A,S; other letters are fairly rare (e.g. Z,J,K,Q,X)
 - **two letters frequencies (e.g. "th" in english)**
 - **most common words**
 - **etc.**
- Cryptoanalysis:
 - **discovered by Arabian scientists in 9th century**
 - **calculate letter frequencies for ciphertext**
 - **compare counts/plots against known values**
 - **have tables of single, double & triple letter frequencies**

Cryptanalysis Example

- given ciphertext:

UZQSOVUOHHXMOVGPOZPEVSGZWSZOPFPESXUBMETSXAIZ
VUEPHZHMDZSHZOWSFAPPDTSPQUZWYMXUZUHSX
EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

- count relative letter frequencies



- guess P & Z are e and t
- guess ZW is th and hence ZWP is the
- proceeding with trial and error finally get:

it was disclosed yesterday that several informal but
direct contacts have been made with political
representatives of the viet cong in moscow

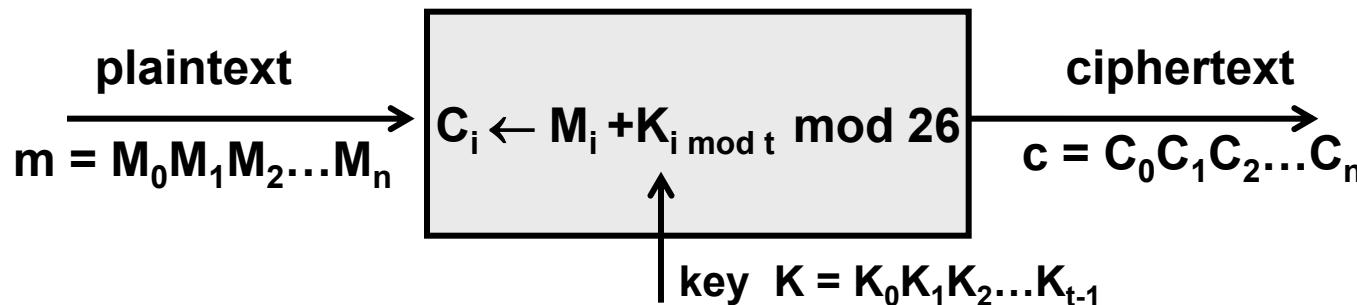


Polyalphabetic Substitution Ciphers

- An approach to improve security is to use multiple cipher alphabets
 - **polyalphabetic substitution ciphers**
 - **uses a set monoalphabetic substitutions**
 - **defines a rule to determine which cipher alphabet (substitution) should be used at each step**
 - normally uses a key to select which substitution is used for each letter of the message
 - repeat from start after end of key is reached

Vigenère Cipher

- Simplest polyalphabetic substitution cipher is the Vigenère Cipher (Blaise de Vigenère, 1523-1596)



- Key is multiple letters long
 - i^{th} letter specifies i^{th} alphabet to use
 - use each alphabet in turn
- Repeat from start after t letters
- Effectively multiple Caesar ciphers
 - $K = K_0 K_1 \dots K_{t-1}$
- Decryption simply works in reverse

0	1	2	3	4	5	6	7	8	9	.	.	.	25
A	B	C	D	E	F	G	H	I	J	K	L	M	N
+ 1	B	C	D	E	F	G	H	I	J	K	L	M	N
+ 2	C	D	E	F	G	H	I	J	K	L	M	N	O
+ 3	D	E	F	G	H	I	J	K	L	M	N	O	P
+ 4	E	F	G	H	I	J	K	L	M	N	O	P	Q
+ 5	F	G	H	I	J	K	L	M	N	O	P	Q	R
+ 6	G	H	I	J	K	L	M	N	O	P	Q	R	S
+ 7	H	I	J	K	L	M	N	O	P	Q	R	S	T
+ 8	I	J	K	L	M	N	O	P	Q	R	S	T	U
+ 9	J	K	L	M	N	O	P	Q	R	S	T	U	V
+ 10	K	L	M	N	O	P	Q	R	S	T	U	V	W
+ 11	L	M	N	O	P	Q	R	S	T	U	V	W	X
+ 12	M	N	O	P	Q	R	S	T	U	V	W	X	Y
+ 13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
+ 14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
+ 15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
+ 16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
+ 17	R	S	T	U	V	W	X	Y	Z	A	B	C	D
+ 18	S	T	U	V	W	X	Y	Z	A	B	C	D	E
+ 19	T	U	V	W	X	Y	Z	A	B	C	D	E	F
+ 20	U	V	W	X	Y	Z	A	B	C	D	E	F	G
+ 21	V	W	X	Y	Z	A	B	C	D	E	F	G	H
+ 22	W	X	Y	Z	A	B	C	D	E	F	G	H	I
+ 23	X	Y	Z	A	B	C	D	E	F	G	H	I	J
+ 24	Y	Z	A	B	C	D	E	F	G	H	I	J	K
+ 25	Z	A	B	C	D	E	F	G	H	I	J	K	L



Vigenère Cipher (cont.)

(00) A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
(01) B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
(02) C D E F G H I J K L M N O P Q R S T U V W X Y Z A B
(03) D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
(04) E F G H I J K L M N O P Q R S T U V W X Y Z A B C D
(05) F G H I J K L M N O P Q R S T U V W X Y Z A B C D E
(06) G H I J K L M N O P Q R S T U V W X Y Z A B C D E F
(07) H I J K L M N O P Q R S T U V W X Y Z A B C D E F G
(08) I J K L M N O P Q R S T U V W X Y Z A B C D E F G H
(09) J K L M N O P Q R S T U V W X Y Z A B C D E F G H I
(10) K L M N O P Q R S T U V W X Y Z A B C D E F G H I J
(11) L M N O P Q R S T U V W X Y Z A B C D E F G H I J K
(12) M N O P Q R S T U V W X Y Z A B C D E F G H I J K L
(13) N O P Q R S T U V W X Y Z A B C D E F G H I J K L M
(14) O P Q R S T U V W X Y Z A B C D E F G H I J K L M N
(15) P Q R S T U V W X Y Z A B C D E F G H I J K L M N O
(16) Q R S T U V W X Y Z A B C D E F G H I J K L M N O P
(17) R S T U V W X Y Z A B C D E F G H I J K L M N O P Q
(18) S T U V W X Y Z A B C D E F G H I J K L M N O P Q R
(19) T U V W X Y Z A B C D E F G H I J K L M N O P Q R S
(20) U V W X Y Z A B C D E F G H I J K L M N O P Q R S T
(21) V W X Y Z A B C D E F G H I J K L M N O P Q R S T U
(22) W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
(23) X Y Z A B C D E F G H I J K L M N O P Q R S T U V W
(24) Y Z A B C D E F G H I J K L M N O P Q R S T U V W X
(25) Z A B C D E F G H I J K L M N O P Q R S T U V W X Y

Example:

key: REBUS (17, 4, 1, 21, 18)

plaintext:

codic emolt osicu ro

key:

REBUS **REBUS** **RE**

ciphertext:

TSECU **VQPFL** **FWJWM** **IS**



Cryptanalysis of Vigenère Ciphers

- Polyalphabetic substitution ciphers make cryptanalysis harder
 - **have multiple ciphertext letters for each plaintext letter**
 - hence letter frequencies are obscured
 - more cipher alphabets to guess and flatter frequency distribution
- But not totally lost
 - **start with letter frequencies**
 - **need to determine number of alphabets (key length)**
 - **then can attack each**



One-Time Pad

- One-Time Pad (OTP) cipher
 - patented by Gilbert Vernam, Bell Telephone Laboratories, in 1919
 - first described by Frank Miller (a California banker) in 1882
- Can be seen as a special case of Vigenère cipher
 - the key $k=\{K_0, K_1, K_2, \dots, K_n\}$ is as long as the plaintext m
 - a random key k is used for each message
- Ciphertext contains no statistical relationship to the plaintext
 - for any plaintext and any ciphertext there exists a key mapping one to other
- In case of alphabet $\{0,1\}$, it is: $c = m \text{ XOR } k$
 - for each bit: $c_i = m_i \text{ XOR } k_i$
 - two possible substitutions:
 - $\{0,1\} \rightarrow \{0,1\}$, with $k_i=0$
 - $\{0,1\} \rightarrow \{1,0\}$, with $k_i=1$



One-Time Pad (cont.)

- Example in hexadecimal:

$m = 48656c6c6f2c207468697320697320616e206578616d706c65$

$k = 159535d62ed94961c45b5019d2717f0ab74c614549e3c1911d$

$c = 5df059ba41f56915ac322339bb025f6bd96c043d288eb1fd78$

Example in binary:

$m = 0100100001100101011011000110110001101111001011000010000001110100 \dots$

$k = 0001010110010101001101011101011000101110110110010100100101100001 \dots$

$c = 01011101111000001011001101110100100000111101010110100100010101 \dots$



One-Time Pad (cont.)

- Claude Shannon (1945) introduced the definition of perfect secrecy and demonstrated that the one-time pad achieves that level of security
 - **the cipher will be unconditionally secure (unbreakable)**
 - no statistical relationship between distinct ciphertexts
 - for any plaintext of equal length to the ciphertext, there is a key that produces that plaintext
- Disadvantages:
 - **can only use the key once**
 - requires a key stream as long as the sum of all messages that has to be encrypted
 - possible problems on distributing and store this long key



Transposition Ciphers

- Transposition is another classical encryption technique
 - hides the message by rearranging the letter order (blocks of bits)
 - without altering the actual letters used
 - performs a sort of permutation
- Can recognize these since have the same frequency distribution as the original text
- Example
 - permutation



Example: Row Transposition Ciphers

- Write letters of message out in rows over a specified number of columns
- then reorder the columns according to some key before reading off the rows

Key: 4 3 1 2 5 6 7

Plaintext: a t t a c k p
 o s t p o n e
 d u n t i l t
 w o a m x y z

Ciphertext: TTNAAPMTSUOAODWCOIXKNLYPETZ



Product Ciphers

- Ciphers using substitutions or transpositions may be not sufficiently secure
- Hence consider using several ciphers in succession to make harder:
 - **two substitutions make a more complex substitution**
 - **two transpositions make a more complex transposition**
 - **a substitution followed by a transposition make a new much harder cipher**
- This is bridge from classical to modern ciphers



Rotor Machines

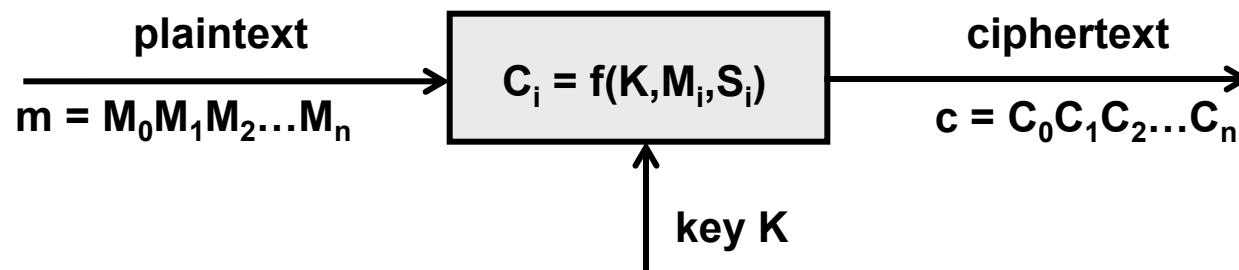
- Before modern ciphers, rotor machines were most common product cipher
- Were widely used in WW2
 - German Enigma, Allied Hagelin, Japanese Purple
- Enigma uses a series of cylinders, each giving one substitution, which rotated and changed after each letter was encrypted
 - every key press caused one or more rotors to step by one
 - implements a varying substitution cipher
 - polyalphabetic substitution cipher
- With 3 cylinders have $26 \times 26 \times 26 = 26^3 = 17576$ alphabets



Stream and Block Ciphers

Stream ciphers

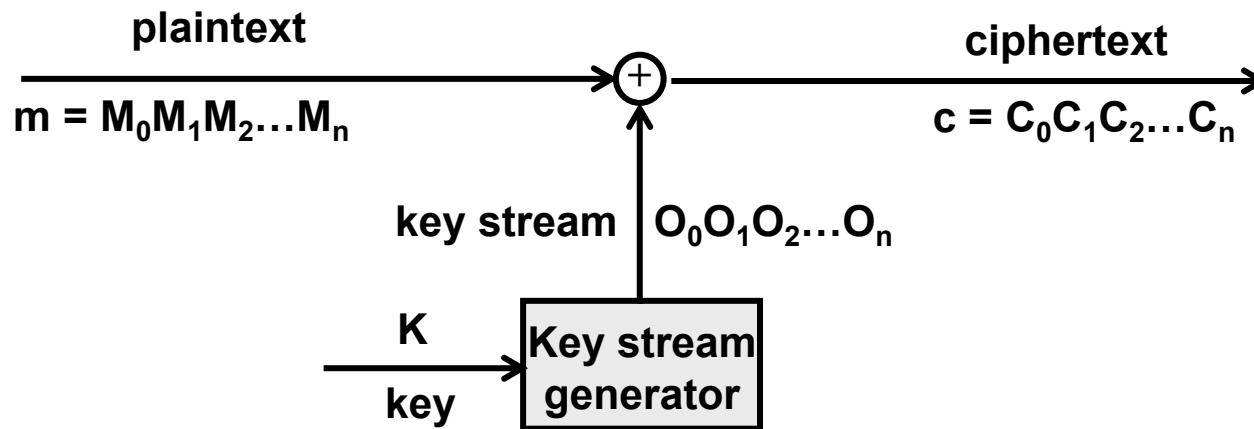
- There are two basic cipher structures
 - Stream ciphers
 - Block ciphers
- Stream ciphers process messages (Encryption/Decryption) a bit or byte at a time when en/decrypting



- The output unit C_i may be function of the current input M_i , an internal state S_i , and the secret key K
 - the internal state S_i may be a function of previous M_j and/or C_j , with $j < i$

Stream ciphers (cont.)

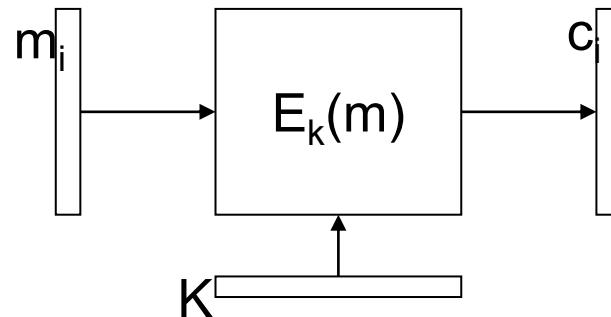
- A very common scheme for stream ciphers (autokey ciphers) is:



- Ideally want a key as long as the message (OTP)
- Most stream ciphers are based on pseudorandom number generators (PRNG)
 - the key is used to initialize the generator, and either key bytes or plaintext bytes are fed back into the generator to produce the byte stream

Block ciphers

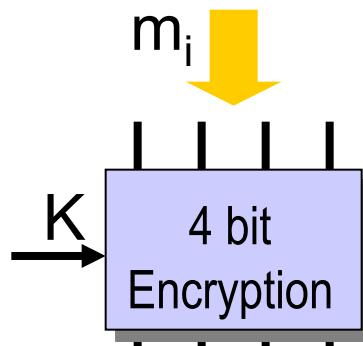
- Block ciphers process messages into blocks, each of which is then en/decrypted
 - **plaintext and ciphertext are treated as a sequence of n-bit blocks of data**
 - **ciphertext is same length as plaintext**
 - **ciphertext depends on plaintext and a key value**



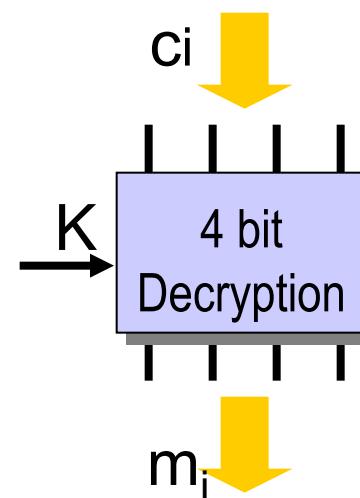
- Larger messages must be processed into blocks, each of which is then en/decrypted
 - **the resulted cipher can be made to behave as a stream cipher**
 - **e.g. CBC, OFB, CFB, and CTR modes**
- Many current ciphers are block ciphers (DES, IDEA, AES, etc.)

Block ciphers (cont.)

- Same input blocks encrypted with the same key are transformed into the same output block
- Example: block size = 4 bits



Plaintext	Ciphertext
0000	1110
0001	0100
0010	1101
0011	0001
0100	0010
0101	1111
0110	1011
0111	1000
1000	0011
1001	1010
1010	0110
1011	1100
1100	0101
1101	1001
1110	0000
1111	0111



Ciphertext	Plaintext
0000	1110
0001	0011
0010	0100
0011	1000
0100	0001
0101	1100
0110	1010
0111	1111
1000	0111
1001	1101
1010	1001
1011	0110
1100	1011
1101	0010
1110	0000
1111	0101



Block ciphers (cont.)

- Like a substitution on (very big) set of possible inputs
 - If the block size is n bits, 2^n possible input values are mapped to 2^n output values
 - like a n -bit substitution
 - A way for encrypting could be to specify completely the mapping table (substitution table)
 - permutation of 2^n n -bit inputs
 - there are $2^n!$ different possible transformations
 - If $n=64$, would need table of 2^{64} entries storing 64-bit blocks
 - $2^{64} \text{ 2}^6 \text{ bit} = 2^{70} \text{ bit} = 2^{67} \text{B} = 2^{37} \text{TB} \approx 10^{11} \text{TB}$
 - it is too long
- Instead, a block cipher is created from smaller building blocks and a secret key
 - if n is the cipher size and k is the key length, there are a total of 2^k possible transformations, rather than 2^n !



Block cipher (cont.)

- How long should the plaintext block be?
 - **having block size too small**
 - in case of known-plaintext attack, an opponent may try to collect $\{M_i, C_i\}$ pairs and construct a decryption table
 - n-bit block cipher requires 2^n pairs
 - It is not required to find the key
 - In case of ciphertext-only attack, if a sequence of M_i have some properties (recognizable sequences of plaintext), it is possible to cryptanalyze the sequence of C_i
 - e.g. exploitation of language redundancy
 - **having block size too long, it could be inconvenient due to the increasing of complexity**
- 64-bit or 128-bit blocks are often used
 - **it is difficult to obtain all 2^{64} pairs (known-plaintext attack)**



Product ciphers

- Product cipher is a type of block cipher that works by executing in sequence a number of simple transformations such as substitution, permutation, and modular arithmetic
- Usually consist of iterations of several rounds of the same algorithm
 - while the individual operations are not themselves secure, it is hoped that a sufficiently long chain would generate sufficient confusion and diffusion as to make it resistant to cryptanalysis
- The operation must be reversible
- Important sub-classes of product ciphers are:
 - Feistel ciphers
 - SP-networks



Design principles of product ciphers

- Product ciphers are defined in terms of:
 - **Block size**
 - increasing size improves security, but slows cipher
 - **Key size**
 - increasing size improves security, makes exhaustive key searching harder, but may slow cipher
 - **Number of rounds**
 - increasing number improves security, but slows cipher
 - **Subkey generation**
 - greater complexity can make analysis harder, but slows cipher
 - **Round function**
 - greater complexity can make analysis harder, but slows cipher
 - **Performances**
 - fast software en/decryption & ease of analysis



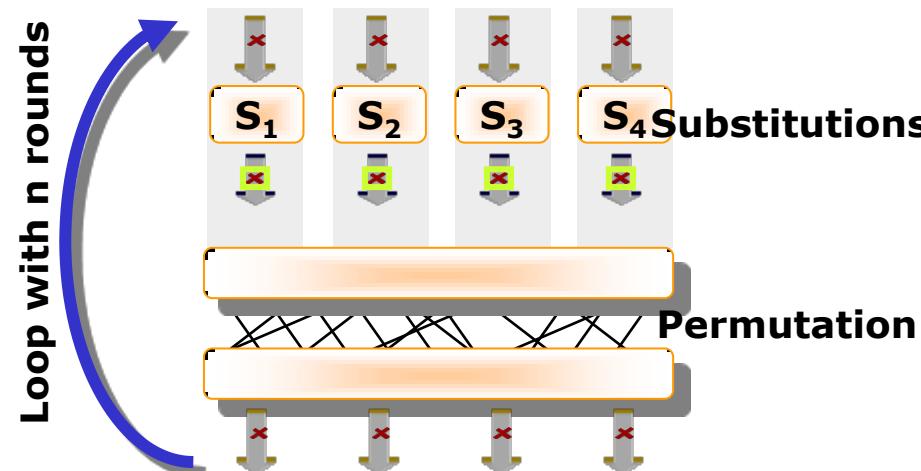
Avalanche Effect

- Key desirable property of encryption algorithms
 - **where a change of one input or key bit results in changing approx half output bits**
 - Avalanche effect
- This is a characteristic of block ciphers
 - **Stream cipher can just have a forward (avalanche) effect**
 - if XOR is used, just the corresponding ciphertext bit is affected
- Modern block ciphers exhibit strong avalanche effect

Advanced Encryption Standard (AES)

Substitution–permutation network

- SP-network is a product cipher that uses only substitutions and permutations
- One possible way to build a block cipher based on SP-network is
 - break the input into managed-sized chunks (say 8 bits),
 - do a substitution on each small chunk,
 - and then take the output of all the substitutions and run them through a permuter (big as the input)
 - the process is repeated, so that each bit winds up as input to each substitution
 - each time is called *round*





Advanced Encryption Standard (AES)

- Block cipher designed to replace DES
 - organized by National Institute of Standards and Technology (NIST)
 - NIST standard on November 26, 2001
 - FIPS PUB 197 (FIPS 197)
 - chosen from five candidate algorithms
 - reviewed by US government (NSA), industry and academia
 - required a four-year process to pick the algorithm
 - winning algorithm chosen Oct 2, 2000
 - also known as Rijndael block cipher
 - original name of the algorithm submitted to AES selection process
 - developed by Joan Daemen and Vincent Rijmen (Belgium)
 - currently, one of the most popular algorithms used in symmetric key cryptography
 - also adopted as an encryption standard by the U.S. government



Advanced Encryption Standard (AES) (cont.)

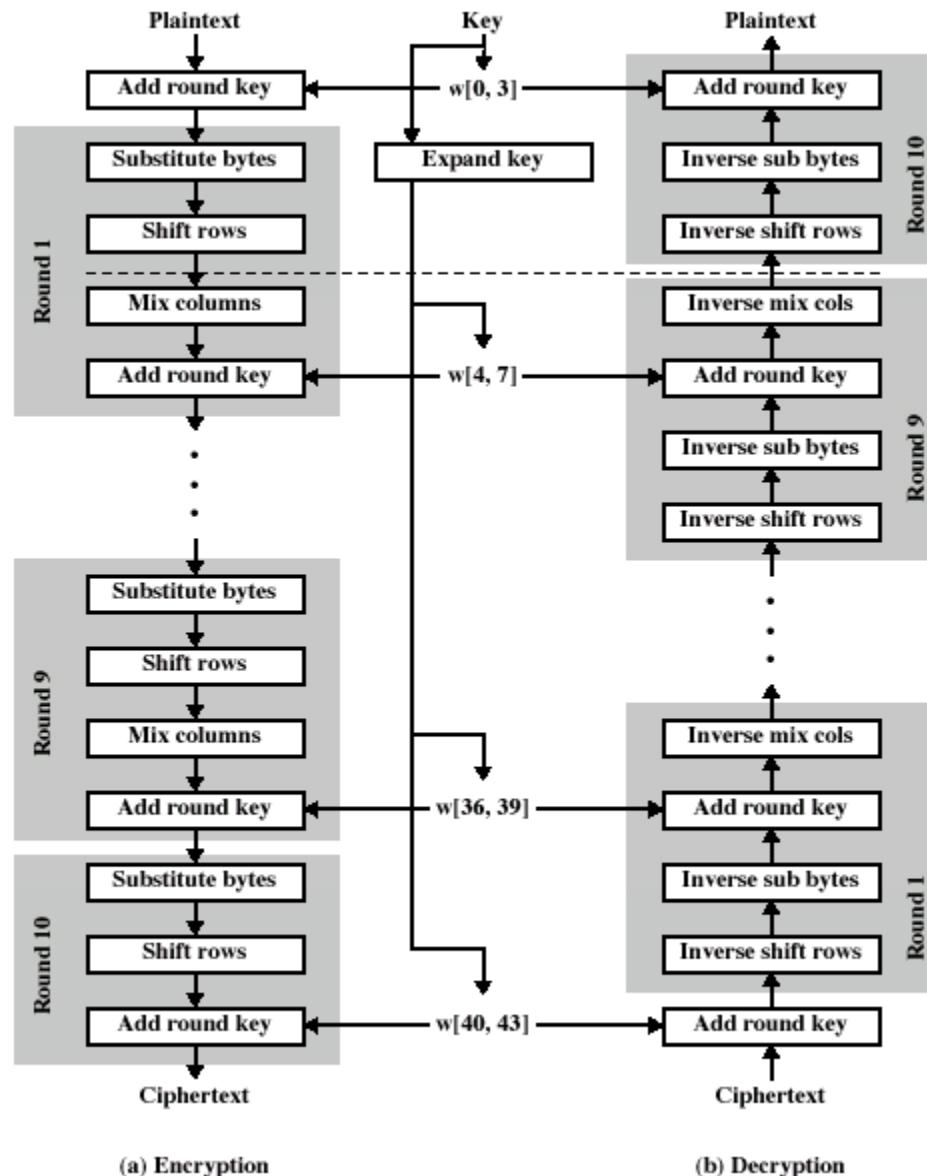
- AES is not precisely the original Rijndael
 - **the original Rijndael algorithm supports a larger range of block and key sizes**
 - Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits
- AES has fixed block size of 128 bits and a key size of 128, 192, or 256 bits
 - **i.e. block size: 16 bytes, key size: 16, 24, or 32 bytes**
- Unlike DES, Rijndael is a substitution-permutation network, not a Feistel network
- Fast in both software and hardware
 - **relatively easy to implement**
 - **requires little memory**



AES Description

- Due to the fixed block size of 128 bits, AES operates on a 4×4 array of bytes, termed the state
 - **v**ersions of Rijndael with a larger block size have additional columns in the state
- AES has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys
- Most AES calculations are done in a special finite field

AES cipher



(a) Encryption

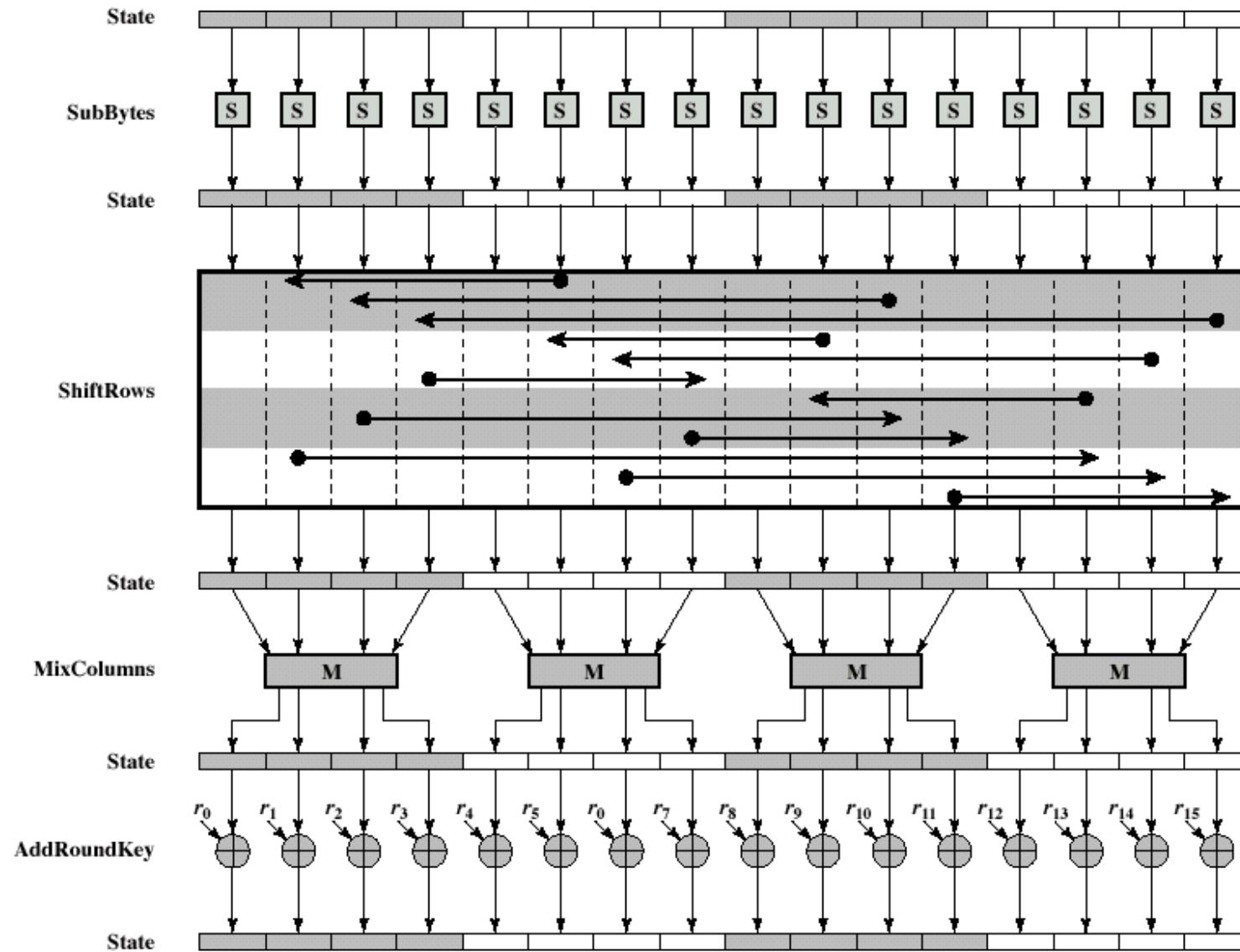
(b) Decryption



AES cipher

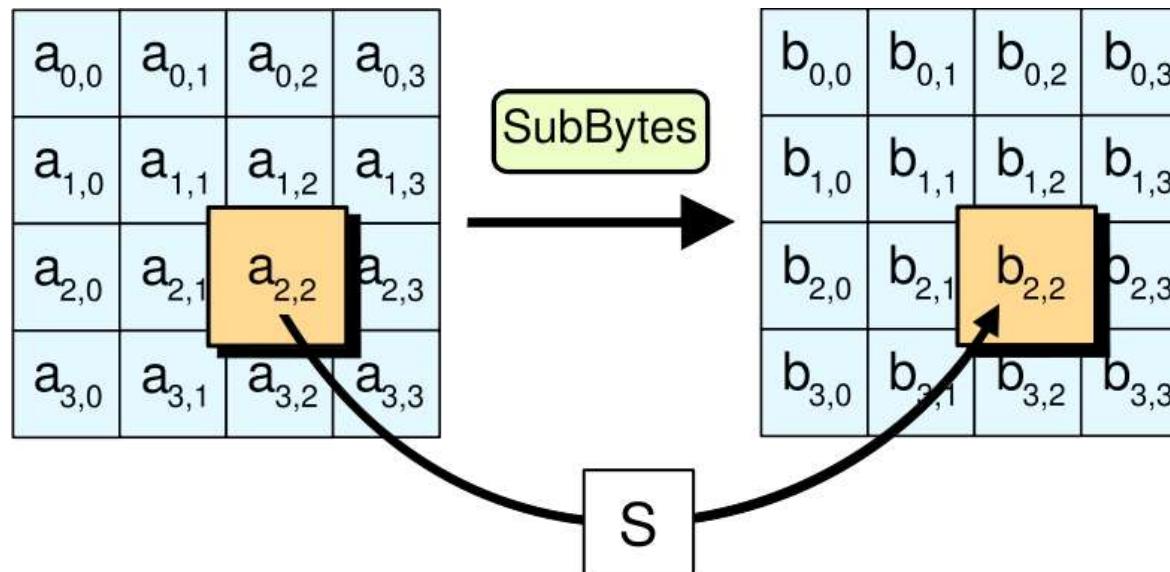
- Algorithm
 - KeyExpansion using Rijndael's key schedule
 - Initial Round
 - AddRoundKey
 - (N-1) Rounds (9 for 128bit key, 11 for 224bit key, 13 for 256bit key)
 1. SubBytes — a non-linear substitution step where each byte is replaced with another according to a lookup table
 2. ShiftRows — a transposition step where each row of the state is shifted cyclically a certain number of steps
 3. MixColumns — a mixing operation which operates on the columns of the state, combining the four bytes in each column
 4. AddRoundKey — each byte of the state is combined with the round key derived from the cipher key using a key schedule
 - Final Round (no MixColumns)
 1. SubBytes
 2. ShiftRows
 3. AddRoundKey

AES single round



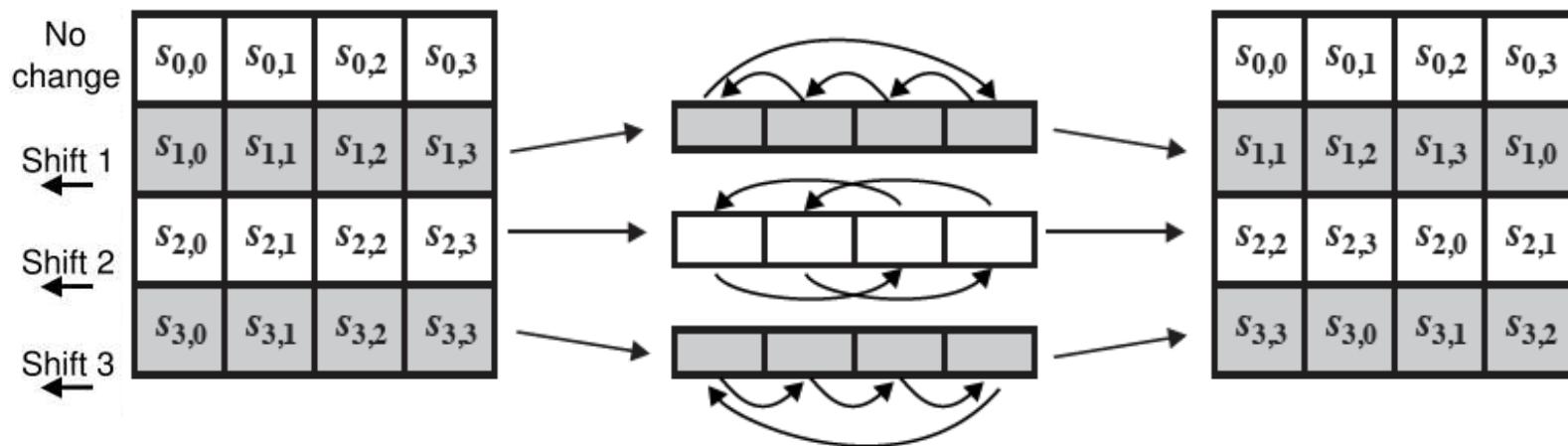
AES SubBytes step

- Each byte in the state is replaced using an 8-bit substitution box, the Rijndael S-box ($b_{ij} = S(a_{ij})$)
 - **S-box can be represented by a table of 256 8-bit values**
 - the transformation is a simple table lookup
 - **the S-box is derived from the multiplicative inverse over GF(2⁸), known to have good non-linearity properties**
 - this operation provides the non-linearity in the cipher



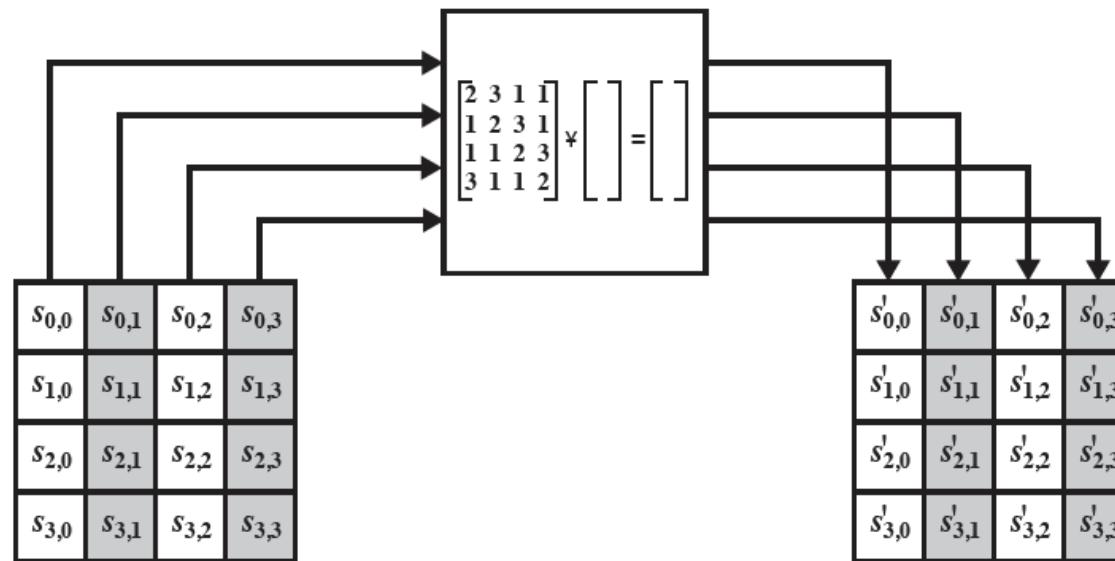
AES ShiftRows step

- ShiftRows step operates on the rows of the state
 - it cyclically shifts the bytes in each row by a certain offset
 - for AES, the first row is left unchanged
 - each byte of the second row is shifted one to the left
 - similarly, the third and fourth rows are shifted by offsets of two and three respectively



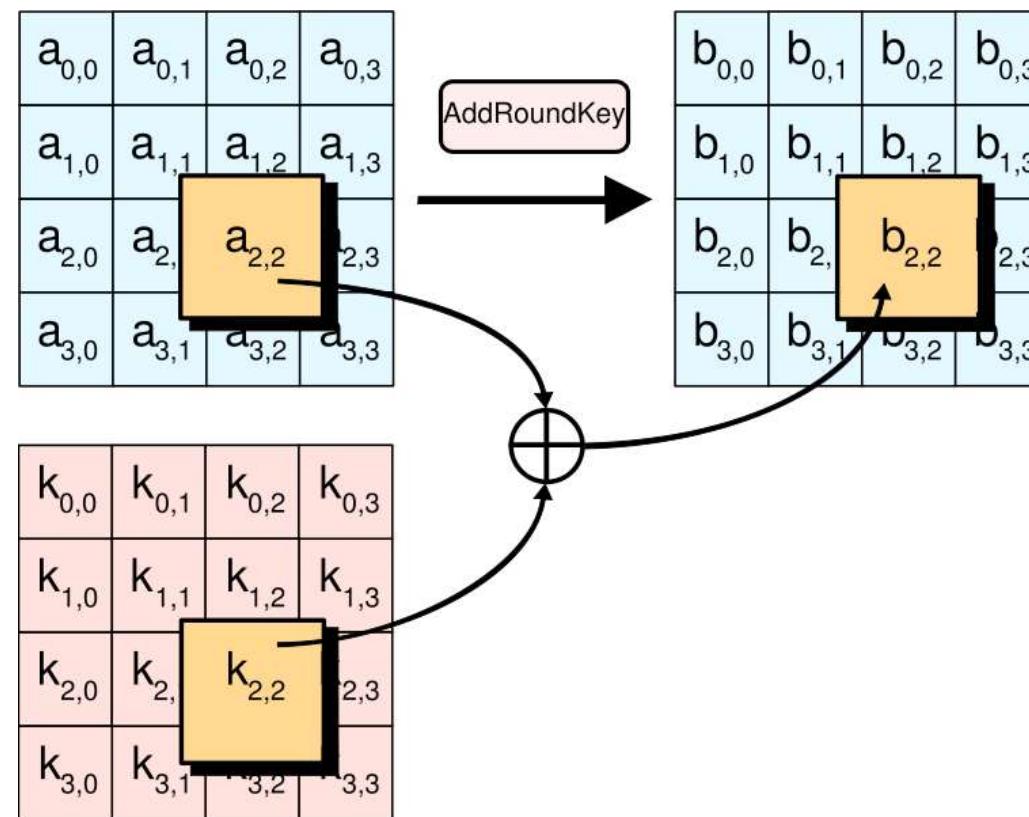
AES MixColumns step

- Four bytes of each column of the state are combined using an invertible linear transformation
 - each byte of column is a function of all four bytes in that column
 - can be viewed as a multiplication $S' = C \times S$ of a particular matrix C by the state S
 - each column i ($i=0,1,2,3$) is treated as a polynomial over $GF(2^8)$ and is then multiplied modulo $x^8+x^4+x^3+x+1$ with an polynomial $c_i(x)$
 - together with ShiftRows, it provides diffusion in the cipher



AES AddRoundKey step

- The subkey is combined (XORed) with the state
 - for each round, a subkey with same size as the state is derived from the main key (key schedule algorithm)
 - the subkey is added to the state using bitwise XOR





AES Security

- In June 2003, the US Government announced that AES may be used also for classified information
 - **this marks the first time that the public has had access to a cipher approved by NSA for encryption of TOP SECRET information**
- The first key-recovery attacks on full AES were published in 2011
 - **the attack is faster than brute force by a factor of about four**
 - it requires $2^{126.1}$ operations to recover an AES-128 key
 - it requires $2^{189.7}$ operations to recover an AES-192 key
 - it requires $2^{254.4}$ operations to recover an AES-256 key
- The only successful attacks against AES have been side channel attacks
 - **however they require a lot of physical information from the system that executes the algorithm**



AES Security (cont.)

- Brute force attack
 - **AES key sizes**
 - 128-bit key
 - $2^{128} = 3.4 \times 10^{38}$ possible keys
 - 192-bit key
 - $2^{192} = 6.2 \times 10^{57}$ possible keys
 - 256-bit key
 - $2^{256} = 1.1 \times 10^{77}$ possible keys
 - Comparing to DES, If you could crack a DES key in one second (i.e., try 2^{56} keys per second), it would take 149 trillion years to crack a 128-bit AES key at the same speed
 - **the universe is believed to be less than 20 billion years old**
 - **but, things change**

**Building a stream cipher using a
block cipher
(Block cipher modes)**



Encrypting large messages

- Block ciphers encrypt fixed size blocks
 - eg. AES has fixed block size of 128 bits and a key size of 128, 192, or 256 bits
- Usually we have arbitrary amount of information to encrypt (longer than the block size)
 - need way to use in practice
- Five modes have been defined for TDEA in: *NIST Special Publication 800-38A* (2001)
 - Electronic Code Book (ECB)
 - Cipher Block Chaining (CBC)
 - (k-bit) Output Feedback (OFB)
 - (k-bit) Cipher Feedback (CFB)
 - Counter Mode (CTR)
- These schemes are applicable to any block cipher
- Other modes are also possible

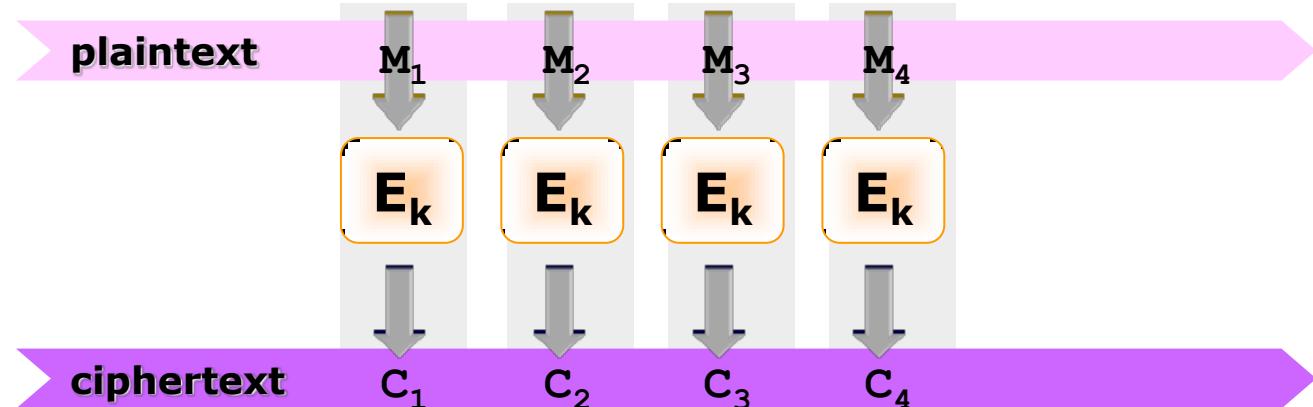


Padding

- If the encryption mode requires that the length of the whole message has to be a multiple of a given block size, a Padding operation has to be performed first
- Examples of padding algorithms:
 - **Bit padding (e.g. ISO/IEC 7816-4)**
 - symbol '1' (bit) is added, and then as many '0' bits as required are added
 - **ANSI X.923**
 - the last byte defines the number of padding bytes; the remaining bytes are filled with zeros
 - eg. [b1 b2 b3 00 00 00 00 05] (3 data bytes, then 5 bytes pad+count)
 - **RFC 5652 Cryptographic Message Syntax / PKCS#7 / PKCS#5**
 - the value of each added byte is the number of bytes that are added
 - eg. [b1 b2 b3 05 05 05 05 05]

Electronic Codebook (ECB) Mode

- Consist of doing the obvious thing, and it is usually the worst method
- The message is broken into n blocks of b with padding for the last one
 - $b = \text{block size of the cipher}$
 - $n \cdot b = \text{message size including padding}$
- Each block is independently encrypted with the secret key K
 - $C_i = E_K(M_i)$
 - $M_i = D_K(C_i)$



- Each block is a value which is substituted, like a codebook
- The cipher text has the same size of the plaintext (excluding padding)

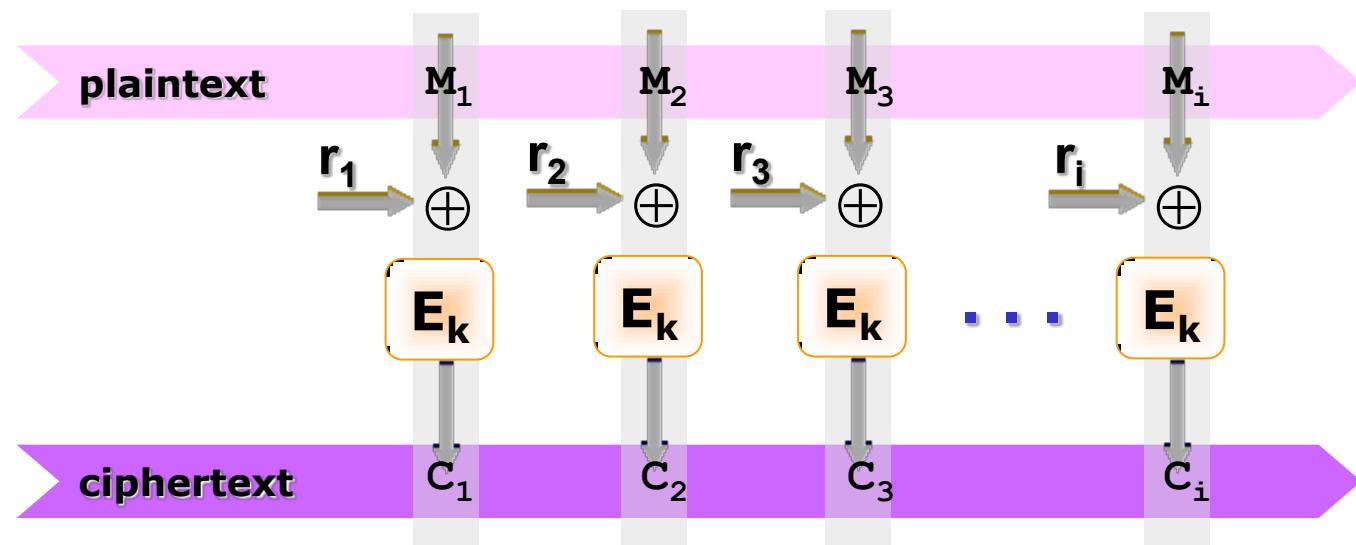


Advantages and Limitations of ECB

- ECB is very simple and does not introduce extra operation (except for padding)
- There are a number of problems that arise and that don't show up in the single block case
 - **repetitions in message may show in ciphertext if aligned with message block**
 - if a message contains 2 identical blocks, the corresponding cipher blocks are identical; it can be a problem
 - in some cases it can be possible to guess a portion of the message
 - **by comparing two ciphertexts it is possible to discover similarities in the plaintexts**
 - no avalanche effect
 - **(partially) knowing the plaintext, is possible to rearrange the ciphertext blocks in order to obtain a new (known) plaintext**
- As result, ECB is rarely used
 - **main use is sending a few blocks of data (e.g. a secret key)**

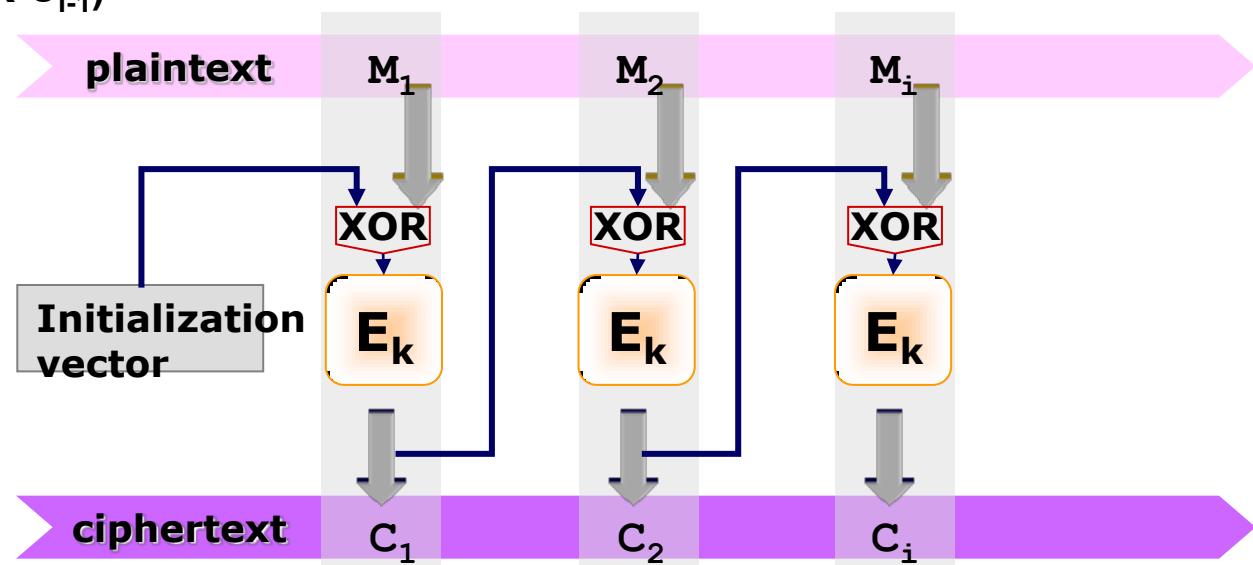
Cipher Block Chaining (CBC) Mode

- CBC objective: if the same block repeats in the plain text, it will not cause repeats of ciphertext
 - avoiding some problems in ECB
- How: adds a feedback mechanism to the cipher
- Result: plaintext is more difficult to manipulate
- Basic idea:



Cipher Block Chaining (CBC) Mode

- Plaintext patterns are concealed by XORing a block of m with the previous block of c
- Requires an IV (Initialization vector) of random data to encrypt the first block
 - $C_i = E_k(M_i \text{ XOR } C_{i-1})$
 - $C_0 = \text{IV}$



- Decryption is simple because \oplus is reversible ($A=B\oplus C \Leftrightarrow A\oplus C=B$):
 - $M_i = D_k(C_i) \text{ XOR } C_{i-1}$



Advantages and Limitations of CBC

- Each ciphertext block depends on **all** previous message blocks
 - **thus a change in the message affects all ciphertext blocks after the change as well as the original block**
 - forward avalanche effect
 - **a change in IV changes all bits of the ciphertext**
- CBC has the same performance of ECB, except for the cost of generating and transmitting the IV (and the cost of one \oplus)
- It can be used a constant value as IV (e.g. 0), however it can lead to some problems
 - **e.g. if a message is transmitted periodically, it is possible to guess if changes occurred**
 - **if the value is known, attackers may supply chosen plaintext**



CBC Threat 1 - Modifying ciphertext blocks

- Using CBC does not eliminate the problem of someone modifying the message in transit
- If IV is sent in the clear, an attacker can change bits of the first plaintext block (working on the ciphertext and IV) by simply changing the IV
 - changing bit *h* of IV has predictable effect to bit *h* of M_1 ,
 - hence either IV must be a fixed value or it must be sent encrypted in ECB mode before rest of message
 - e.g. by changing $\text{IV} \rightarrow \text{IV}' = \text{IV} \oplus X$, when decrypting we have:
$$\text{M}_1' = D_K(C_1) \oplus \text{IV}' = (\text{M}_1 \oplus \text{IV}) \oplus \text{IV}' = \text{M}_1 \oplus \text{IV} \oplus \text{IV} \oplus X = \text{M}_1 \oplus X$$
- If the attacker changes the ciphertext block C_i , C_i gets \oplus 'd with the decrypted C_{i+1} to yield M_{i+1}
 - since $M_{i+1} = D_K(C_{i+1}) \oplus C_i$
 - changing bit *h* of C_i has predictable effect to bit *h* of decrypted M_{i+1}
 - however the attacker cannot know the new decrypted M_i' (a new random block, as side effect)
 - e.g. by changing $C_i \rightarrow C_i' = C_i \oplus X$, when decrypting we have:
$$\text{M}_i' = D_K(C_i') \oplus C_{i-1} = D_K(C_i \oplus X) \oplus C_{i-1} = \text{unpredictable without knowing } K$$
$$\text{M}_{i+1}' = D_K(C_{i+1}) \oplus C_i' = (\text{M}_{i+1} \oplus C_i) \oplus C_i' = \text{M}_i \oplus X$$



CBC Threat 2 - Rearranging ciphertext blocks

- Knowing the plain text, the corresponding ciphertext and IV, it is possible to rearrange the C_1, C_2, C_3, \dots (building blocks), in such a way to obtain a new known $M'_1, M'_2, M'_3 \dots$
 - e.g. suppose an intruder rearranges the plaintext $c=C_1, C_2, C_3, C_4$:
 - if the modified ciphertext $c'=C_1, C_3, C_2, C_4$, then when decrypting:
$$M'_1 = D_K(C_1') \oplus IV' = D_K(C_1) \oplus IV = M_1$$
$$M'_2 = D_K(C_2') \oplus C_1' = D_K(C_3) \oplus C_1 = (M_3 \oplus C_2) \oplus C_1$$
$$M'_3 = D_K(C_3') \oplus C_2' = D_K(C_3) \oplus C_3 = (M_3 \oplus C_2) \oplus C_3$$
$$M'_4 = D_K(C_4') \oplus C_3' = D_K(C_4) \oplus C_3 = M_4$$
 - if the modified ciphertext $c'=C_1, C_3, C_2, C_4$, then when decrypting:
$$M'_1 = D_K(C_1') \oplus IV' = D_K(C_1) \oplus IV = M_1$$
$$M'_2 = D_K(C_2') \oplus C_1' = D_K(C_3) \oplus C_1 = (M_3 \oplus C_2) \oplus C_1$$
$$M'_3 = D_K(C_3') \oplus C_2' = D_K(C_2) \oplus C_3 = (M_2 \oplus C_1) \oplus C_3$$
$$M'_4 = D_K(C_4') \oplus C_3' = D_K(C_4) \oplus C_2 = (M_4 \oplus C_3) \oplus C_2$$
- These threads can be combated by adding a MIC (message integrity check) code, or a strong checksum to the plaintext before encrypt
 - use of 32 bit checksum doesn't completely solve the problem, since 1 in 2^{32} chance that the checksum will work
 - after 2^{31} attempts, the probably of obtaining a correct checksum is ~50%



Output Feedback (OFB) Mode

- Acts like a pseudorandom number generator
 - more precisely as a stream cipher based on XOR
- The message is encrypted by \oplus ing it with the pseudorandom stream generated by the OFB
 - message is treated as a stream of bits
- How it works:
 - A pseudorandom number O_0 is generated (named IV as in CBC)
 - O_0 is encrypted (using secret key K) obtaining O_1
 - from O_1 is obtained O_2 and so on, as many block are needed
$$O_i = E_K(O_{i-1})$$
$$O_0 = IV$$
 - the pseudorandom stream (like a one-time pad) is independent of message and can be computed in advance
 - the one-time pad is simply \oplus 'd with the message
$$C_i = M_i \text{ XOR } O_i$$
- Example of uses: stream encryption over noisy channels

Output Feedback (OFB) Mode

- OFB in short:

- a long pseudorandom string is generated (one-time pad)

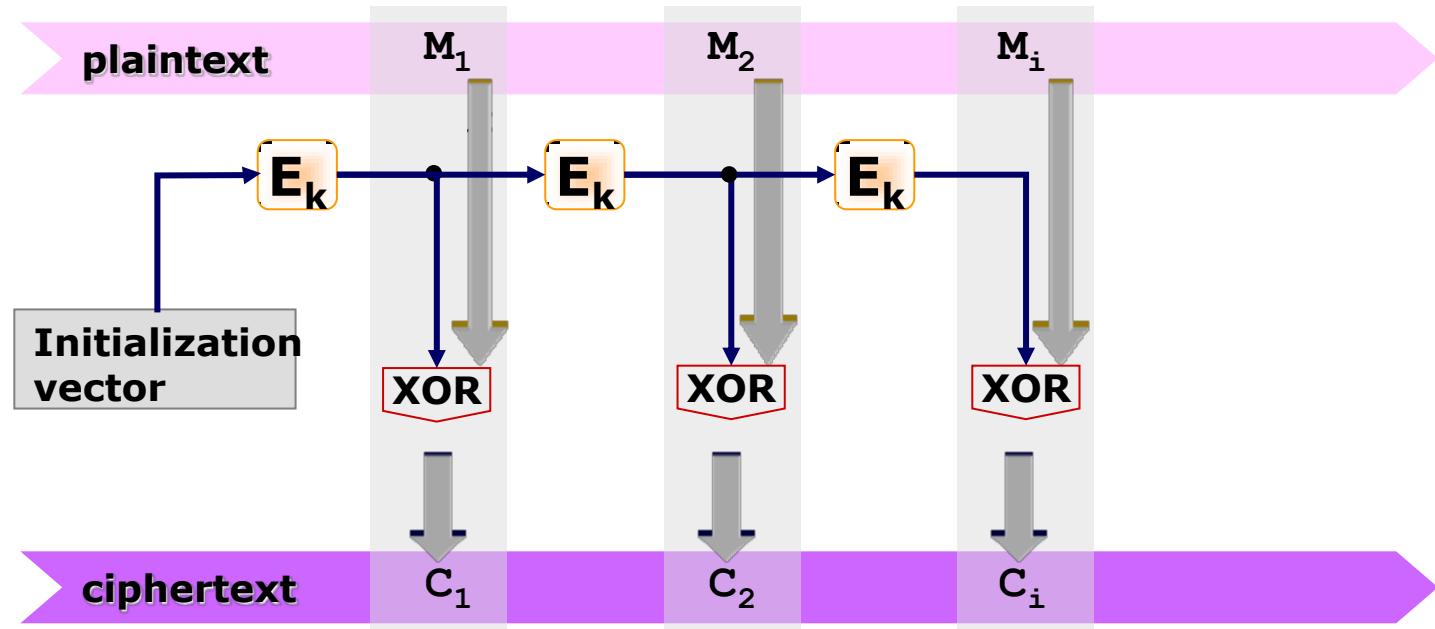
$$O_i = E_K(O_{i-1})$$

$$O_0 = IV$$

- the one-time pad is \oplus 'd with the message

$$C_i = M_i \text{ XOR } O_i$$

$$M_i = C_i \text{ XOR } O_i$$





Advantages and Limitations of OFB

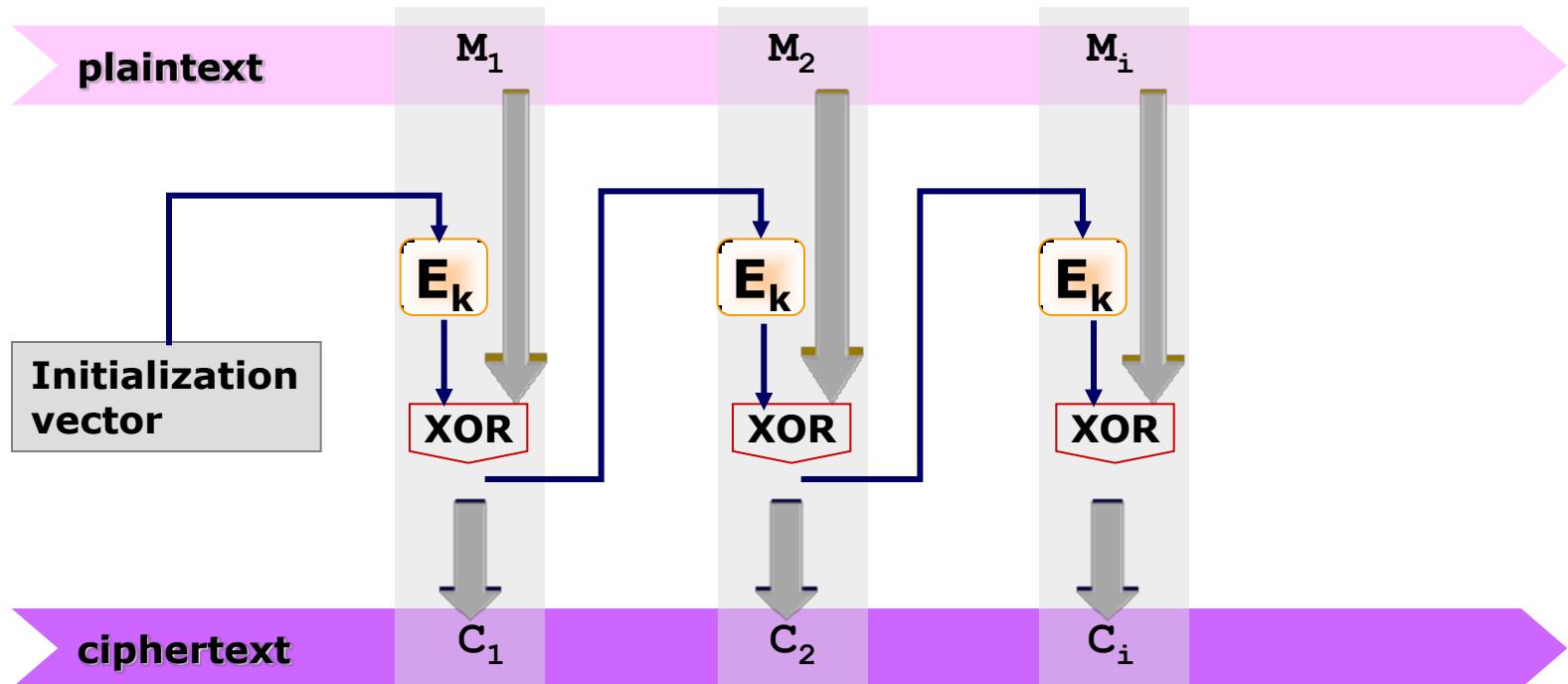
- Advantages of OFB:
 - one-time pad can be generated in advance
 - if a message arrives in arbitrary-sized chunks, the associated ciphertext can immediately be transmitted
 - possibility to encrypt variable-length messages: no need of padding
 - if some bits of the ciphertext get garbled, only those bits of plaintext get garbled (no error propagation)
- Disadvantages of OFB:
 - if the plaintext is known by a bad guy, he can modify the ciphertext forcing the plaintext into anything he wants
 - hence must never reuse the same sequence (key+IV)
 - sender and receiver must remain in sync, and some recovery method is needed to ensure this occurs

Cipher Feedback (CFB) Mode

- Similar to OFB
- The b bits shifted in to the encryption module are the b bits of the ciphertext from the previous block

$$C_0 = \text{IV}$$

$$C_i = M_i \text{ XOR } E_K(C_{i-1})$$





Advantages and Limitations of CFB

- The block cipher is used in encryption mode at both ends
 - like in OFB
 - however:
 - the one-time pad cannot be generated entirely in advance
 - needs to stall while do block encryption after every b bits
- Errors in cipher text propagate to the next block
 - like in CBC
- The lost of a portion of the ciphertext can be resumed if it is multiple of s-bit used by the CFB
 - it is possible to have s-bit CFB with s different from b (i.e. the $E_k(\cdot)$ size), e.g. 8 bits
 - with OFB or CBC if octects (bytes) are lost in transmission or extra octects are added, the rest of transmission is garbled
 - with 8-bit CFB as long as an error is an integral number of octects, things will be resynchronized

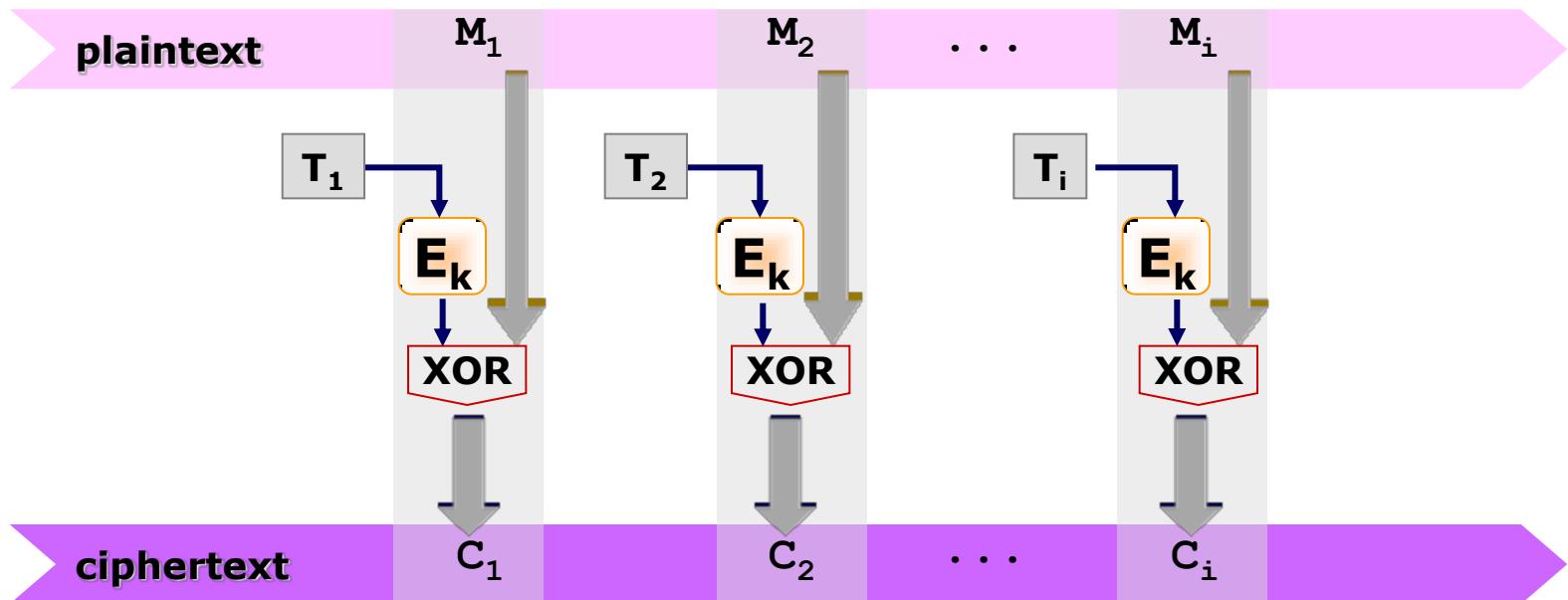
Counter (CTR) Mode

- Similar to the OFB
- The CTR output blocks are generated by encrypting a set of input blocks T_i called counters

$$O_i = E_K(T_i)$$

$$C_i = M_i \text{ XOR } O_i$$

➤ the sequence of counters T_1, T_2, \dots, T_i must have the property that each block in the sequence is different from every other block





Counter (CTR) Mode (cont.)

- Typically, the counter is initialized to some value and then incremented by 1 for each subsequent block (modulo 2^n , where n is the block size)
- Advantages:
 - the forward cipher functions can be applied to the counters prior to the availability of the plaintext or ciphertext data
 - like OFB
 - the cipher functions can be performed in parallel
 - the plaintext block that corresponds to any particular ciphertext block can be recovered independently from the other plaintext blocks if the corresponding counter block can be determined



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Hash functions

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Hash Function

- Also known as Message Digest
- It is a function that takes a variable-length input message and produces a fixed-length output

$$h = H(m)$$

- **input message m of any size**
- **output data h of fixed size**
- **the output h is called message digest**
- The transformation $H(m)$ is one-way
 - **it is not practical to figure out which input corresponds to a given output**
- Examples: MD5, SHA-1, SHA-2



Hash function properties

- Compression
 - **It reduces data size, “summarizing” the characteristics of the message**
 - input message of variable size
 - output of fixed size
 - is function of the entire input message
 - » allows the detection of possible modifications/errors
- Pseudorandomness
 - **the value of message digest should look “randomly generated”**
- Fast calculation (efficiency)
 - **given an input x , $h(x)$ is easy (fast) to compute**
 - requires low processing resources



Hash function properties (cont.)

- Def: given a value h , if $H(x) = h$ then x is called preimage of h
- Def: given a pair x, y with $x \neq y$, if $H(x) = H(y)$, we have a collision
- One-way (or preimage resistance)
 - **for any output, it is computationally infeasible to find any input which hashes to that output**
 - given a value h (for which m is unknown), find m' such that $H(m') = h$
- Weak collision resistance (or second-preimage resistance)
 - **it is computationally unfeasible to find any second input which has the same output as any specified input**
 - given m , find $m' \neq m$ such that $H(m') = H(m)$
- (Strong) collision resistance
 - **it is computationally unfeasible to find any two distinct inputs m, m' which hash to the same output**
 - find m and m' such that $H(m') = H(m)$



How many bits should the output have?

- How many bits should the output have in order to prevent someone from being able to find a collision?
- If the message digest has n bits, then it would take (expected value) $2^{n/2}$ messages chosen at random (Birthday Paradox)
 - **this is why message digest functions should have output of at least 160 or 256 bits (in place of just 128 as for symmetric cryptography)**
- However sometime it is not sufficient for an attacker to find out just two messages with the same hash
 - **in such case, a brute-force attack requires 2^n searches (mean value 2^{n-1})**
 - similarly to a brute force attack to a symmetric cipher



Preimage vs collision attack complexity (1/2)

- Preimage brute force attack complexity
 - **n = hash size**
 - **N=2ⁿ is the number of possible different hash values**
 - **P(k)= Pr{success with k tries} = Pr{success with k input messages}**
 - $P(1) = 1/N = 1 - (1-1/N)$
 - $P(2) = 1 - (1-1/N)^2$
 - $P(3) = 1 - (1-1/N)^3$
 - $P(k) = 1 - (1-1/N)^k$
since $(1-x)^k \approx 1-kx$ when $x \ll 1$, then:
 - $P(k) \approx 1 - (1-k/N) = k/N$
 - **Look for the value of k such that $P(k) \geq 50\%$**
 - $P(k) \geq 1/2$
 - $k/N \geq 1/2$
 - $k \geq N/2 = 2^{n-1}$
- Same complexity of a brute force attack against a symmetric cipher secret key



Preimage vs collision attack complexity (2/2)

Collision brute force attack complexity

➤ $P(k) = \Pr\{\text{success with } k \text{ tries}\} = \Pr\{\text{success with } k \text{ input messages}\}$

- $P(2) = 1/N = 1 - (N-1)/N$
- $P(3) = 1 - (N-1)/N * (N-2)/N$
- $P(k) = 1 - (N-1)/N * (N-2)/N * (N-3)/N * \dots * (N-k+1)/N$
 $= 1 - (1-1/N) * (1-2/N) * (1-3/N) * \dots * (1-(k-1)/N)$

➤ Since it is always $1-x \leq e^{-x}$, and if $x << 1$ then $1-x \approx e^{-x}$

$$\begin{aligned} P(k) &\approx 1 - e^{-1/N} e^{-2/N} e^{-3/N} \dots e^{-(k-1)/N} = 1 - \prod_{i=1,..,k-1} e^{-i/N} = e^{-1/N \sum_{i=1,..,k-1} i} = \\ &= 1 - e^{-k(k-1)/2N} \end{aligned}$$

➤ Look for the value of k such that $P(k) \geq 50\%$

- $P(k) > 1/2$
- $e^{-k(k-1)/2N} < 1/2$
- $k(k-1)/2N > \ln(2)$
- $k^2 > 2N \ln(2)$
- $k > \sqrt{N} \sqrt{2 \ln(2)} \approx 1.18 * 2^{n/2}$

● If n bits are sufficient for resisting to preimage brute force attack, $2n$ bits are required to resist to a collision brute force attack

➤ two times the size of a symmetric key resistant to a brute force attack

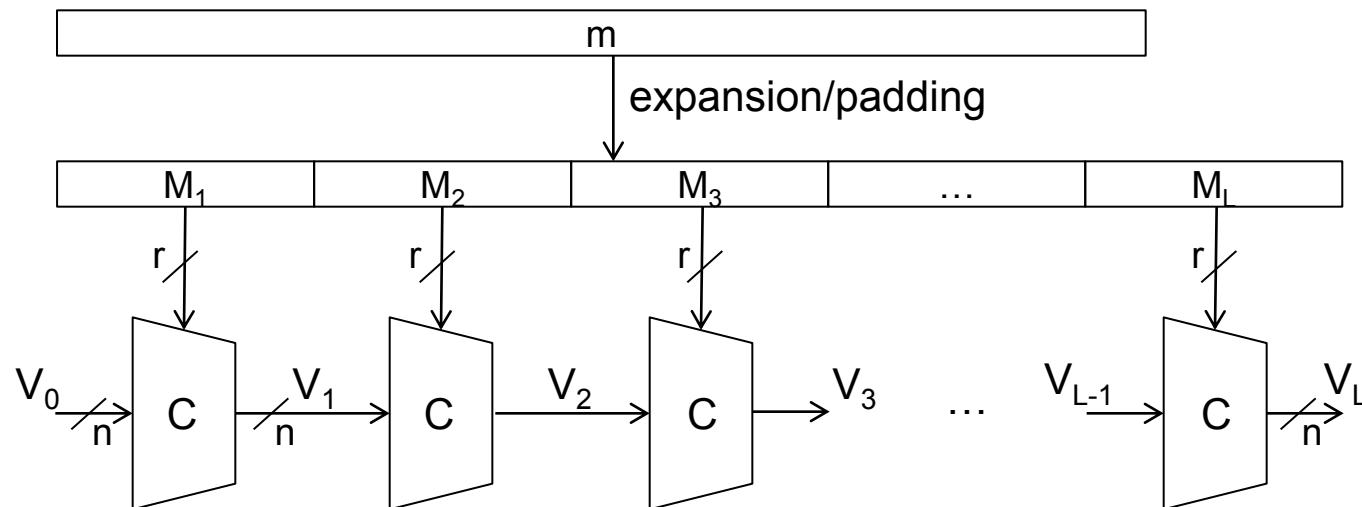


Birthday problem

- Collision on birthday
- Problem: How many people there should be in room in such a way the probability to have at least one birthday collision (two people have the same birthday) is greater than 50%?
- Solution:
 - assuming randomly distributed birthdays, the probability that at least one person is born on a given day is approximately $\approx 23/365 = 0.063 (\approx 6\%)$
 - it is possible to calculate that with 23 people, the probability that at least two people have the same birthday is 0.507 ($\approx 50\%$)
 - using the formula that we already obtained for a general collision attack:
 - $k > 1.18 * \sqrt{365} = 22,54 \Rightarrow k \geq 23$

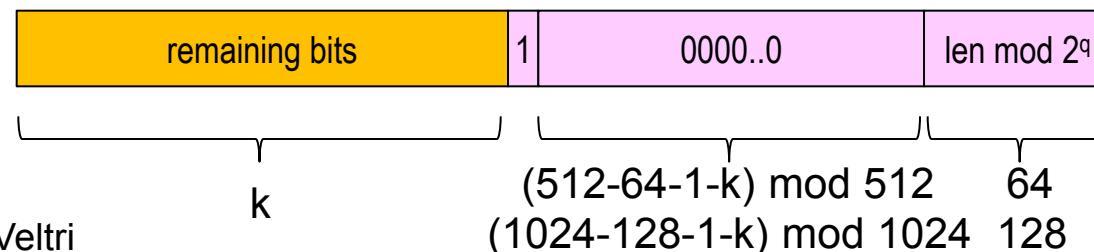
Structure of Hash functions

- Most hash functions H are designed as iterative processes which hash arbitrary length inputs m by processing successive fixed-size blocks of the input
 - Expand m to (M_1, M_2, \dots, M_L) , with a total of $L \cdot r$ bits
 - For $i=1$ to L , compute $V_i = C(V_{i-1}, M_i)$
 - Finally, set $H(m) = h = V_L$
- If appropriate padding is used and compression function C is collision-resistant, then the hash function is collision-resistant (Merkle-Damgard)



Padding

- Message processed in r -bit blocks
 - input message needs to be padded in order to make the total length multiple of r
 - message padding is not required to be invertible
- Example of message padding (used by MD5, SHA)
 - first bit is set to "1"
 - last q bits encode the length of the unpadded message mod 2^q
 - from 0 to up $r-1$ bits set to "0" are added in the middle
 - such that the total length of the padded message is multiple of r
 - e.g.
 - $r=512$, $q=64$
 - $r=1024$, $q=128$





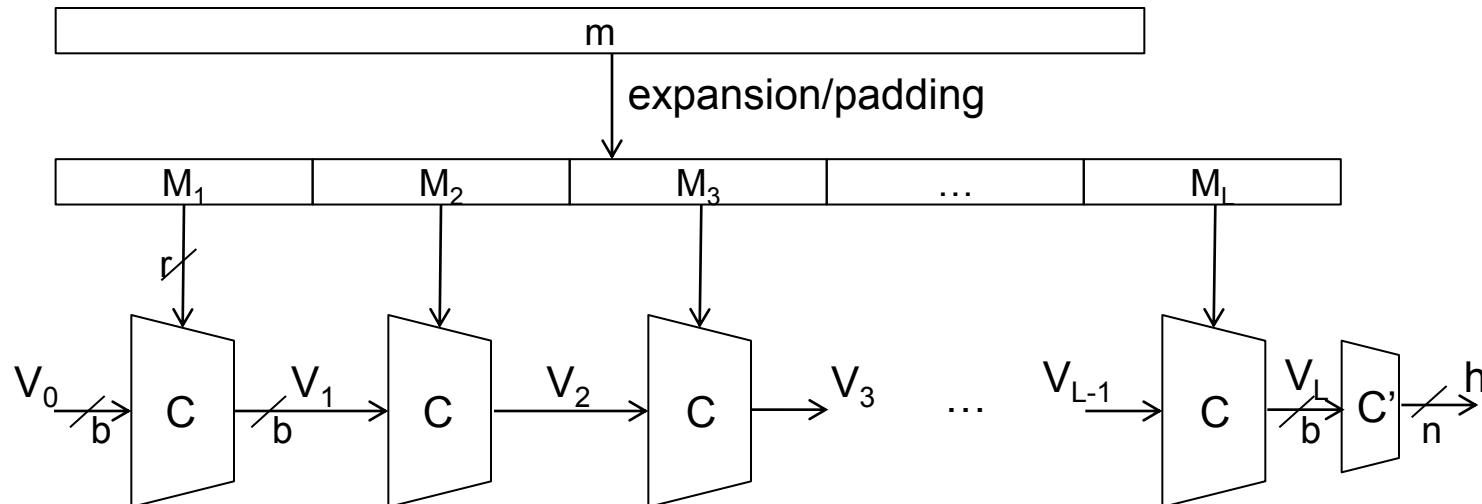
Structure of Hash functions (cont.)

- Possible attack to Merkle-Damgard hash functions:
 - Length extension attack
 - given $h=H(m)$, it is straightforward to compute m' and h' , such that $h'=H(m||m')$, even for unknown m when the padding bits are known, (e.g. in case the padding bits are function of the m length known)
 - the attack is based on using h as an internal hash for computing h'

Structure of Hash functions (cont.)

Wide-Pipe Hash (Stefan Lucks)

- For $i=1$ to L , compute $V_i = C(V_{i-1}, M_i)$, b bits
- Finally, set $H(m) = h = C'(V_L)$, $n < b$ bits





Secure Hash Standard (SHS/SHA)

- Set of cryptographically secure hash algorithms specified by NIST as message digest functions
- The original specification of the algorithm was published in 1993 as the Secure Hash Standard, FIPS PUB 180, by NIST (SHA-0)
- Successively revised by the following standards
 - **SHA-1 (1995)**
 - like SHA-0, it produces a message digest that is 160 bits long
 - **SHA-224, SHA-256, SHA-384, SHA-512 (SHA-2, 2001)**
 - produce digests that are respectively 224, 256, 384, 512 bits long
 - **SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256 (2015)**
 - SHAKE128 and SHAKE256 produce variable length output
- Employed in several widely used security applications and protocols
 - **TLS/SSL, PGP, SSH, S/MIME, IPsec, etc.**



SHA standards

Algorithm	Output size (bits)	Internal state (bits)	Block size (bits)	Word size (bits)	Rounds	Security $O(2^k)$
SHA-2	SHA-1	160	160	512	32	<63 ^(*)
	SHA-224	224	256	512	32	64
	SHA-256	256	256	512	32	64
	SHA-384	384	512	1024	64	80
	SHA-512	512	512	1024	64	80
SHA-3	SHA3-224	224	1600	1152	64	24
	SHA3-256	256	1600	1088	64	24
	SHA3-384	384	1600	832	64	24
	SHA3-512	512	1600	576	64	24
	SHAKE128	any	1600	1344	64	24
	SHAKE256	any	1600	1088	64	<256

(*) SHA1 collision found (Feb, 2017)



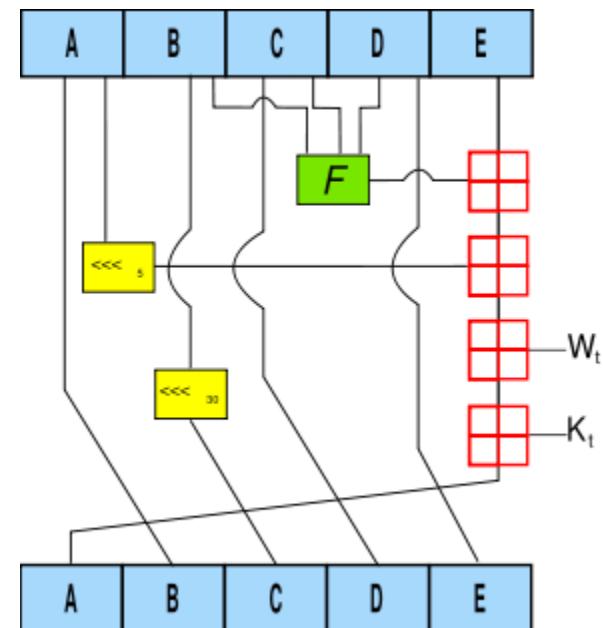
SHA-1

- Secure Hash Standard 1 (SHA-1)
 - **published in 1995**
 - **differs from SHA-0 only by a single bitwise rotation in the message schedule of its compression function**
 - this was done to correct a flaw in the original algorithm which reduced its cryptographic security
- It produces a 160-bit (5 32bit-words) digest
- Based on principles similar to those used by MD5 message
 - **little slower than MD5 and little more secure**
- Operates in stages (as MD5)
 - **processes 512 bit blocks**
 - **uses the same padding mechanism of the MD5**

SHA-1 (cont.)

- Processing of one 512bit block of the input message:

- The 160bit state is view as 5x 32bit words
 - A B C D E
- Each 512bit input block has 16 words
 - $Y = X_0 \ X_1 \dots X_{15}$
- Makes 80 rounds for each input block
- In each round t
 - uses a word W_t derived by X_0, X_1, \dots, X_{15}
 - uses a different constant word K_t out of 4
 - uses a different function F_t out of 4
 - $Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$ $0 \leq t \leq 19$
 - $Parity(x,y,z) = x \oplus y \oplus z$ $20 \leq t \leq 39$
 - $Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$ $40 \leq t \leq 59$
 - $Parity(x,y,z) = x \oplus y \oplus z$ $60 \leq t \leq 79$



$T = \text{ROTL}_5(A) + F_t(B, C, D) + E + K_t + W_t$
 $E = D$
 $D = C$
 $C = \text{ROTL}_{30}(B)$
 $B = A$
 $A = T$
with additions mod 2^{32}

SHA-2

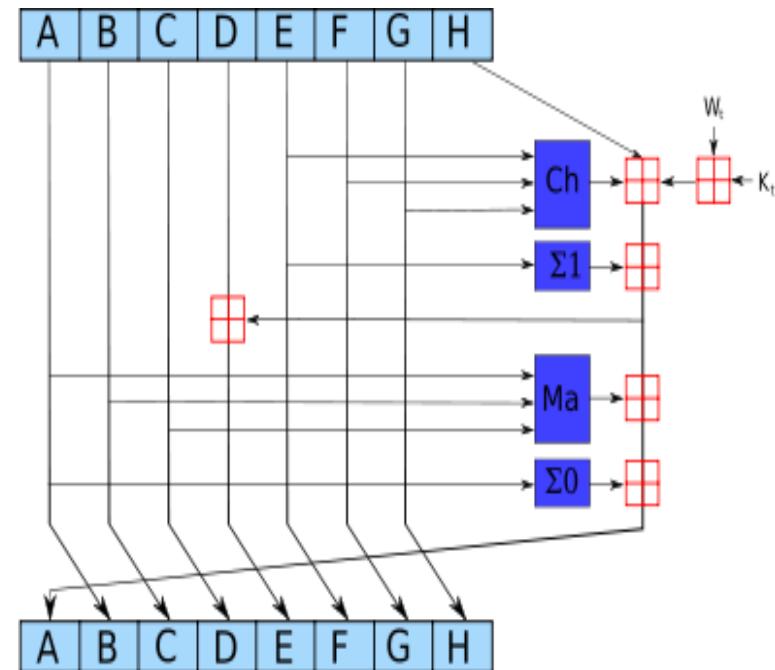
- SHA-224, SHA-256, SHA-384, and SHA-512
- SHA-256 and SHA-512 are computed with 32- and 64-bit words, respectively
 - **use different shift amounts and additive constants**
 - **different number of rounds**
- SHA-224 and SHA-384 are simply truncated versions of the first two, computed with different initial values
- SHA-256 and SHA-512 perform 64 and 80 rounds, respectively

$$\text{Ch}(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$$

$$\text{Maj}(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$$

$$\Sigma_0(X) = \text{ROTR}_{s1}(X) \oplus \text{ROTR}_{s2}(X) \oplus \text{ROTR}_{s3}(X)$$

$$\Sigma_1(X) = \text{ROTR}_{s4}(X) \oplus \text{ROTR}_{s5}(X) \oplus \text{ROTR}_{s6}(X)$$



$$T_1 = H + \Sigma_1(E) + \text{Ch}(E, F, G) + K_t + W_t$$

$$T_2 = \Sigma_0(A) + \text{Maj}(A, B, C)$$

$$H = G$$

$$G = F$$

$$E = D + T_1$$

$$D = C$$

$$C = B$$

$$B = A$$

$$A = T_1 + T_2$$



SHA-3

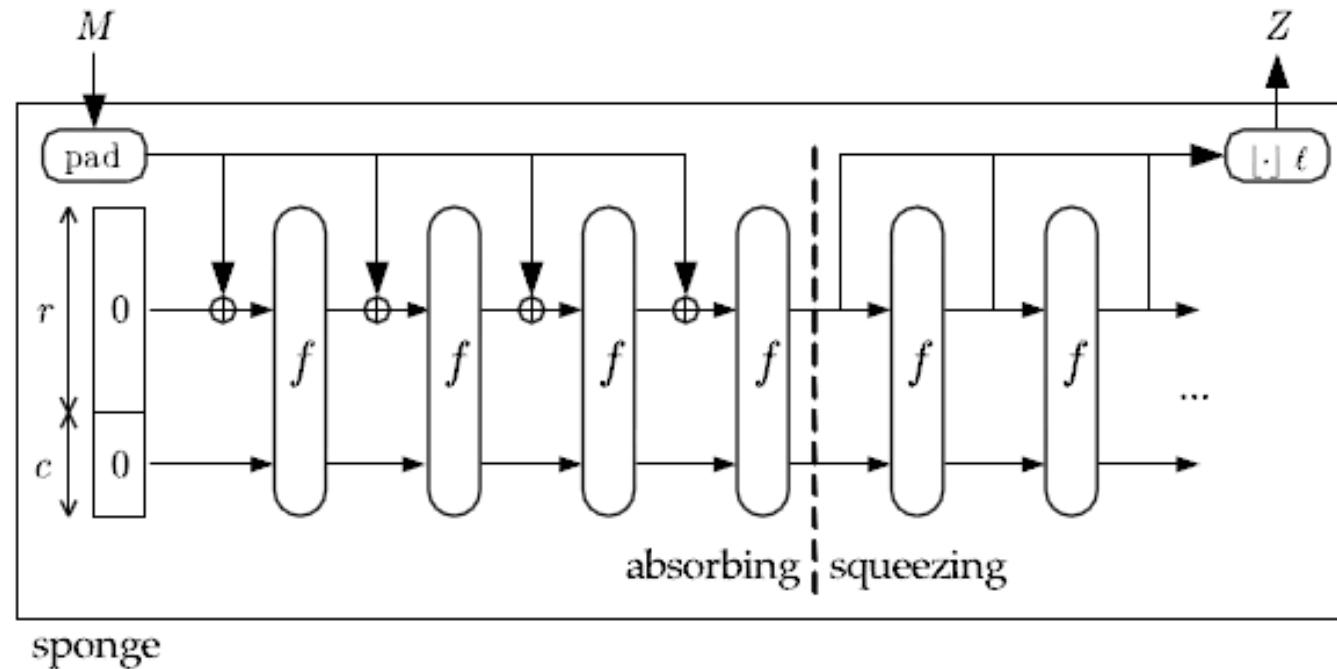
- SHA-1 has been attacked (collision attack)
 - Complexity required for finding a collision is less than 2^{63}
- SHA-2 security is not yet as well-established
 - Not received as much scrutiny as SHA-1
 - Although no practical attacks have yet been reported, SHA-2 is algorithmically similar to SHA-1
- SHA-3
 - Chosen in 2012 after a public competition started by NIST in 2008
 - similar to the development process for AES
 - NIST standard published in 2015
 - Hash function formerly called Keccak
 - It supports the same hash lengths as SHA-2
 - 4 cryptographic hash functions and 2 extendable-output functions
 - Its internal structure differs significantly from the rest of the SHA family. It is a *cryptographic sponge function*



Cryptographic Sponge Functions

- Generalize hash functions to more general functions whose output length is arbitrary
 - **variable-length input variable-length output function based on a fixed length transformation or permutation f operating on a fixed number b of bits (the width)**
 - f operates on a state of $b = r + c$ bits
 - the value r is called the bitrate and the value c the capacity
 - default values for Keccak are $r = 576$ bits, $c = 1024$ bits ($b = 1600$ bits)
 - **it processes blocks in two phases:**
 - the absorbing phase
 - the r -bit input message blocks are XORed into the first r bits of the state, interleaved with applications of the function f
 - the squeezing phase
 - the first r bits of the state are returned as output blocks Z_i , $i=1..k$, interleaved with applications of the function f
 - the number of iterations is determined by the requested number of bits ℓ
 - » $k = \lceil \ell / r \rceil$

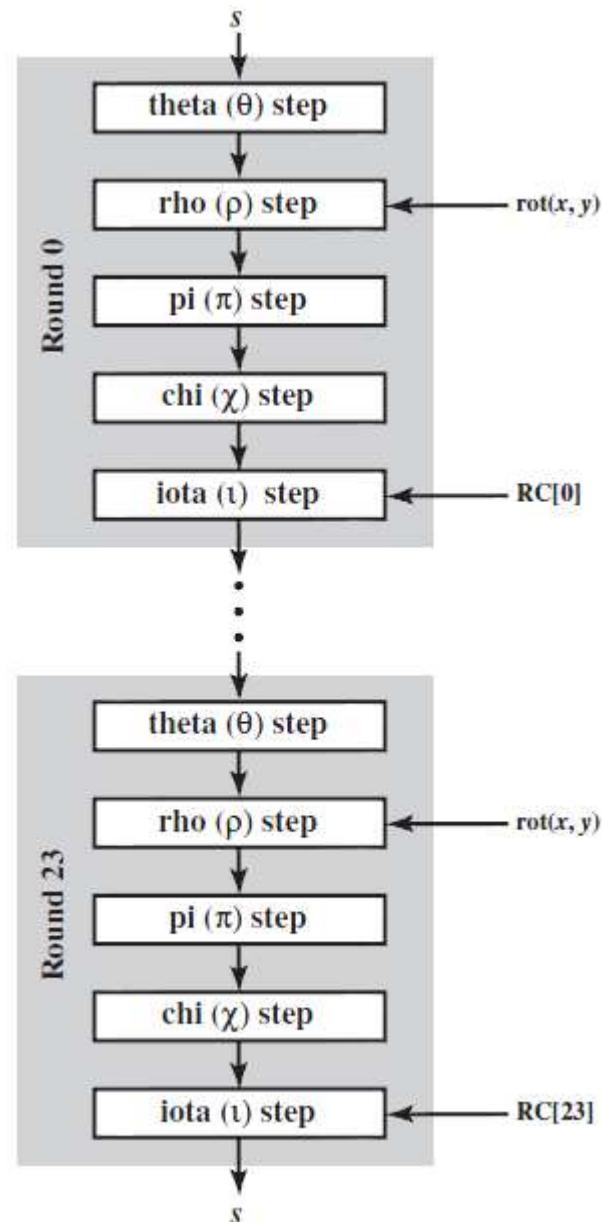
Cryptographic Sponge Functions (cont.)



- If the desired output length ℓ satisfies $\ell \leq b = r + c$, then at the completion of the absorbing phase, the first ℓ bits of the state are returned and the sponge construction terminates otherwise, the sponge construction enters the squeezing phase

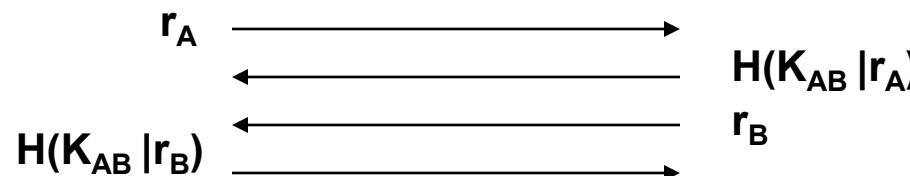
SHA-3 (cont.)

- In SHA-3 that f takes as input a 1600-bit variable s consisting of $b=r+c$ bits
- For internal processing within f , the input/internal state is organized as a 5×5 matrix of 64-bit words (referred as lanes) (1600 bits total)
 - $a[x,y,z]$ is the bit array
 - $L[x, y]$ is the 5×5 matrix
- The basic block permutation function KECCAK- p consists of 24 rounds of processing
 - **each round consists of five steps (functions) denoted by θ , ρ , π , χ , and ι**



What doing with a Hash

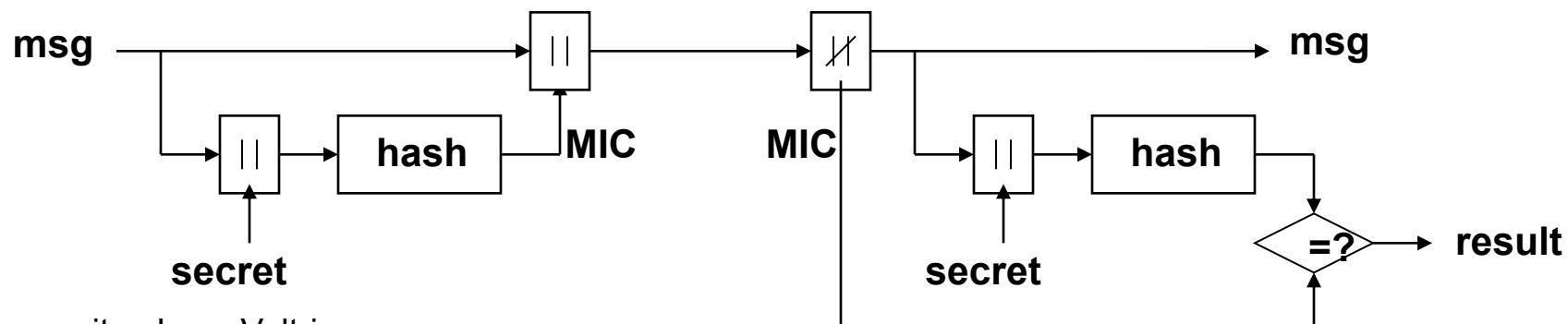
- Message fingerprint
 - integrity check
 - maintaining a copy of a message digest of some data/program in place of the copy of the entire data
- Password Hashing
 - a system may know/store just the hash of a passwd
- Digital signature
 - signing the MD of a message instead of the entire message
 - for efficiency (MDs are easier to compute than public-key algorithms)
- Entity authentication
 - identification



What doing with a Hash (cont.)

Message Authentication

- **$H(m)$ can be used as is a MIC for m , however:**
 - if not protected, can be modified by an intruder (anyone can compute $H(m)$)
 - cannot be used as cryptographic proof of the source
- **possible solutions:**
 - encrypt m and $H(m)$ with a secret key, or
 - compute the hash of both the message m and a secret
 - e.g. $H(k||m)$
 - the result is one-way function that takes two parameters: k and m (MAC function)





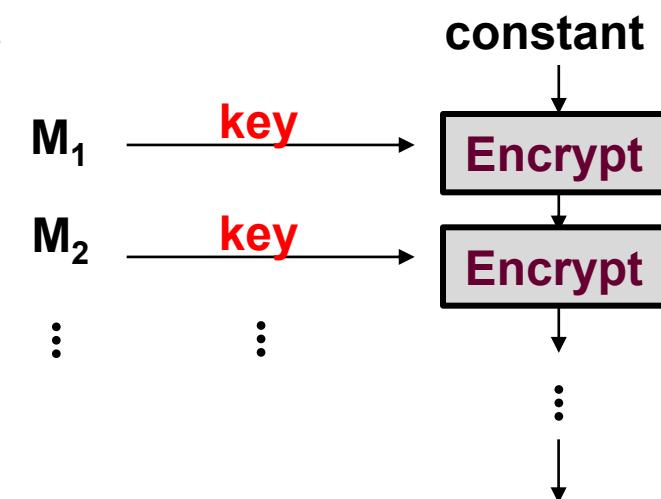
What doing with a Hash (cont.)

● Encryption

- An H function may be also used to build a cipher
- one-time pad
 - just as OFB (and CRT), generating a pseudorandom bit stream and encrypting the message just by a simple \oplus
 - the pseudorandom stream is generated starting from a hash of a secret
 - e.g. $O_1 = H(K_{AB} | IV)$, $O_2 = H(K_{AB} | O_1)$, ..., $O_i = H(K_{AB} | O_{i-1})$
 - same problems as OFB
- mixing in the plaintext
 - as in CFB, the plaintext is mixed in the bit stream generation
 - $B_1 = H(K_{AB} | IV)$, $B_2 = H(K_{AB} | C_1)$, ..., $B_i = H(K_{AB} | C_{i-1})$
 - $C_1 = M_1 \oplus B_1$, $C_2 = M_2 \oplus B_2$, ... , $C_i = M_i \oplus B_i$

Using secret key algorithm for creating a Hash Function

- A hash function can be built by means of a block ciphers
 - the message is padded ad divided in blocks
 - M_1, M_2, \dots, M_n
 - each block is used to encrypt the output of the previous operation
 - $H_i = E_{M_i}[H_{i-1}]$
 - $H_0=0$
 - use final block as the hash value



- Resulting hash can be too small (64-bit)
- Not very fast to compute



Example: Unix password hashing

- The original UNIX password hash "crypt function" uses DES to generate a hash of a password
 - **first convert the passwd (the message) into a “secret key”**
 - the 7bit ASCII codes of the first 8 chars form the 56bit key
 - **the key is used to encrypt the number 0 with a modified DES**
 - 25 DES passes are performed
 - the modified DES is used to prevent HW accelerators designed to DES to be used to reverse the passwd hash
 - the modified algorithm uses a 12-bit random number (salt)
 - **the salt and the final ciphertext are base64-encoded into a printable string stored in the password or shadow file**
- Other Unix/Linux password "crypt" functions have been added; currently they are:
 - **the original DES-based crypt function**
 - **hash-based functions (e.g. MD5-crypt function), where common hash function such as MD5 or SHA-1 are used**
 - such functions generally allow users to have any length password (> 8bytes), and do not limit the password to ASCII (7-bit) text



Example: Unix password hashing (cont.)

- The MD5-crypt function is really not a straight implementation of MD5
 - first the password and salt are MD5 hashed together in a first digest
 - then 1000 iteration loops continuously remix the password, salt and intermediate digest values
 - the output of the last of these rounds is the resulting hash
- A typical output of the stored password together with username, salt, and other information is:

alice:\$1\$BZftq3sP\$xEeZmr2fGEnKjVAxzjQo68:12747:0:99999:7:::

- where \$1\$ indicates the use of MD5-crypt, while BZftq3sP is the base-64 encoding of the salt and xEeZmr2fGEnKjVAxzjQo68 is the password hash



Keyed Hash Functions

- Cryptographic one-way functions that create a small fixed-sized block depending on an input message m and a secret key K
 - $F_k(m) = F(k, m)$
- They condense a variable-length message m to a fixed-sized block
 - often used as message authenticator
 - the result of the function and the function itself are usually referred to as Message Authentication Code (MAC)
- MAC functions are similar to a Hash functions (one-way, collision resistant, etc.)
- The simplest way to build such a function could be to combine an Hash function with the secret key
 - e.g. $F(k, m) \equiv H(m||k)$
- Stronger functions can be designed, like HMAC (RFC 2104)
 - see later



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Introduction to Number Theory and Modular Arithmetic

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



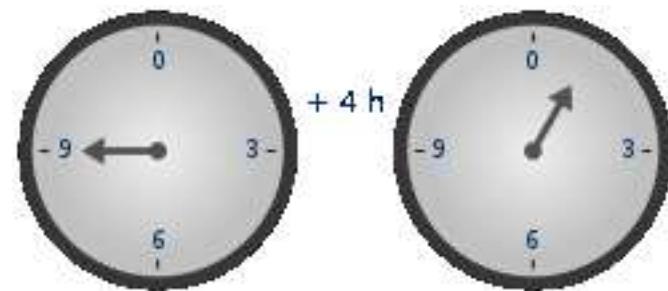
Modular Arithmetic

- Define **modulo operator** $a \bmod n$ to be remainder when a is divided by n
 - $a = qn + r$, where: $q = \text{quotient} = \lfloor a/n \rfloor$ and $0 \leq r \leq n-1$
 - $r = a \bmod n$
- n is called the **modulus**
- The result r of the modulo operation is called the **residue** of $a \bmod n$
- Given two integers a and b , they are said to be “**congruent modulo n** ” if, when divided by n , a and b have same remainder
 - it means that $(a \bmod n) = (b \bmod n)$
 - Use the term **congruence** for: $a \equiv b \pmod{n}$
- Examples
 - $100 \equiv 34 \pmod{11}$
 - $-12 \bmod 7 = -5 \bmod 7 = 2 \bmod 7 = 9 \bmod 7$
- Note:
 - if $a \equiv 0 \pmod{n}$, then $n|a$



Modular Arithmetic (cont.)

- All operations have a result between 0 and $n-1$
- It is “clock arithmetic”
 - uses a finite number of values, and loops back from either end
 - do addition & multiplication and modulo reduce





Modular Arithmetic (cont.)

Properties of congruence:

- $a \equiv b \pmod{n} \Leftrightarrow b \equiv a \pmod{n}$
- $a \equiv b \pmod{n} \Leftrightarrow a-b \equiv 0 \pmod{n} \Leftrightarrow n|(a-b)$
- $\forall k, a \equiv a + kn \pmod{n}$
- $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n} \Rightarrow a \equiv c \pmod{n}$

Properties of addition, subtraction and product:

- $a \underline{\text{op}} b \equiv (a \pmod{n}) \underline{\text{op}} (b \pmod{n}) \pmod{n}$
- i.e. $(a \underline{\text{op}} b) \pmod{n} = ((a \pmod{n}) \underline{\text{op}} (b \pmod{n})) \pmod{n}$
 - with: $\underline{\text{op}} = +, -, *$
- **result: can do reduction at any point, i.e.**
 - $(a+b) \pmod{n} = ((a \pmod{n}) + (b \pmod{n})) \pmod{n}$
- **the property with the product leads also the following result:**
 - $a^k \equiv (a \pmod{n})^k \pmod{n}$, i.e. $a^k \pmod{n} = (a \pmod{n})^k \pmod{n}$

Example of use:

- **$10^3 \pmod{7} = 1000 \pmod{7} = 142 \cdot 7 + 6 = 6$**
 - however it is easier to compute as:
 $10^3 \pmod{7} = (10 \pmod{7})^3 \pmod{7} = 3^3 \pmod{7} = 27 \pmod{7} = 6$



Z_n

- Z_n is defined as the set of all integers ≥ 0 and $< n$
 - $Z_n = \{0, 1, \dots, n-1\}$
 - called **set of residues (mod n)**, or **set of remainders (mod n)**, or **set of residue classes (mod n)**
- Z_n forms a commutative ring for addition with a multiplicative identity element

Property	Expression
Commutative Laws	$(w + x) \text{ mod } n = (x + w) \text{ mod } n$ $(w \times x) \text{ mod } n = (x \times w) \text{ mod } n$
Associative Laws	$[(w + x) + y] \text{ mod } n = [w + (x + y)] \text{ mod } n$ $[(w \times x) \times y] \text{ mod } n = [w \times (x \times y)] \text{ mod } n$
Distributive Law	$[w \times (x + y)] \text{ mod } n = [(w \times x) + (w \times y)] \text{ mod } n$
Identities	$(0 + w) \text{ mod } n = w \text{ mod } n$ $(1 \times w) \text{ mod } n = w \text{ mod } n$
Additive Inverse ($-w$)	For each $w \in Z_n$, there exists a z such that $w + z \equiv 0 \text{ mod } n$



Example – Arithmetic modulo 8

Addition modulo 8

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Multiplication modulo 8

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Additive and multiplicative
inverses modulo 8

w	$-w$	w^{-1}
0	0	—
1	7	1
2	6	—
3	5	3
4	4	—
5	3	5
6	2	—
7	1	7



Divisors

- Say a non-zero number b **divides** a if for some m have $a=mb$
(a, b, m all integers)
 - **that is b divides into a with no remainder**
 - **denote this $b \mid a$**
 - **and say that b is a divisor of a**
- Example
 - **all of 1,2,3,4,6,8,12,24 divide 24**



Prime Numbers

- Prime numbers only have divisors of 1 and self
 - **they cannot be written as a product of other numbers**
 - **note: 1 is prime, but is generally not of interest**
- e.g. 2,3,5,7 are prime, 4,6,8,9,10 are not
- Prime numbers are central to number theory
- List of prime numbers less than 200 is:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89
97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179
181 191 193 197 199



Prime Factorization

- To **factor** a number n is to write it as a product of other numbers:
 $n = a \times b \times c$
- Note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
- The **prime factorization** of a number a is when it's written as a product of primes
 - eg. $91 = 7 \times 13$; $3600 = 2^4 \times 3^2 \times 5^2$

$$a = \prod_{p \in P} p^{a_p}$$



Relatively Prime Numbers

- Two numbers a, b are **relatively prime** if have **no common divisors** apart from 1
- Example
 - 8 and 15 are relatively prime since factors of 8 are 1,2,4,8 and of 15 are 1,3,5,15 and 1 is the only common factor



Greatest Common Divisor (GCD)

- A common problem in number theory
- GCD of a and b , that is $\text{GCD}(a,b)$, is the largest number that divides both a and b
 - eg $\text{GCD}(60,24) = 12$
- The greatest common divisor can be determined by comparing their prime factorizations and using least powers
 - Example
 - $300 = 2^1 \times 3^1 \times 5^2$ $18 = 2^1 \times 3^2$ hence $\text{GCD}(18, 300) = 2^1 \times 3^1 \times 5^0 = 6$
- Often want **no common factors** (except 1)
 - hence numbers are relatively prime
 - e.g. $\text{GCD}(8,15) = 1$
 - 8 & 15 are relatively prime



Euclid's GCD Algorithm

- An efficient way to find the $\text{GCD}(a,b)$
- Uses theorem that:
 - $\text{GCD}(a,b) = \text{GCD}(b, a \bmod b)$
 - **dim**
 - if $d|a,b$ (d divides a and b), then $a=h*d$ and $b=k*d$ where h,k are the quotients, then $a \bmod b = r_a = a - q_a b = hd - q_a kd = (h-q_a k)d \Rightarrow d|(a \bmod b)$
 - moreover, starting from $a = r_a + q_a b$, if $d|b$ and $d|r_a$, with $r_a=s*d$, then $a = r_a + q_a b = (s+q_a k)d \Rightarrow d|a$
 - **so, common divisors of a,b are also common divisor of b and r_a**



Euclid's GCD Algorithm (cont.)

- If we use it several times:

➤ $\text{GCD}(a,b) = \text{GCD}(b, a \bmod b) = \text{GCD}(b, r_1) = \text{GCD}(r_1, b \bmod r_1) = \text{GCD}(r_1, r_2) = \text{GCD}(r_2, r_1 \bmod r_2) = \text{GCD}(r_2, r_3) = \dots = \text{GCD}(r_n, 0) = r_n$

where:

- $r_1 = a \bmod b$
- $r_2 = b \bmod r_1$
- $r_3 = r_1 \bmod r_2$
- \dots

➤ $r_k = r_{k-2} \bmod r_{k-1}$

➤ the formula is valid also for r_1 and r_2 if we define:

- $r_0 = b, r_{-1} = a$

➤ note: at each step, $r_k < r_{k-1}$

➤ calculate r_k using r_{k-1} and r_{k-2} , until $r_{n+1} = r_{n-1} \bmod r_n$ is equal to 0, then

➤ $\text{GCD}(a,b) = \text{GCD}(r_n, 0) = r_n$



Euclid's GCD Algorithm (cont.)

- Euclid's Algorithm to compute $GCD(a,b)$:

```
A←a, B ← b
while B>0 {
    R ← A mod B
    A ← B
    B ← R
}
return A
```



Example GCD(1970,1066)

gcd(1970, 1066)	1970 = 1 × 1066 + 904
gcd(1066, 904)	1066 = 1 × 904 + 162
gcd(904, 162)	904 = 5 × 162 + 94
gcd(162, 94)	162 = 1 × 94 + 68
gcd(94, 68)	94 = 1 × 68 + 26
gcd(68, 26)	68 = 2 × 26 + 16
gcd(26, 16)	26 = 1 × 16 + 10
gcd(16, 10)	16 = 1 × 10 + 6
gcd(10, 6)	10 = 1 × 6 + 4
gcd(6, 4)	6 = 1 × 4 + 2
gcd(4, 2)	4 = 2 × 2 + 0
gcd(2, 0)	



Multiplicative inverse (modulo n)

- The multiplicative inverse of a number x is the number we multiply x by to get 1
 - with real numbers this is just $1/x$
 - the multiplicative inverse of $m \text{ mod } n$ is $u : u^*m = 1 \text{ (mod } n)$
 - u^*m differs from 1 by a multiple of n , or
$$u^*m + v^*n = 1$$
- The Extended Euclid's Algorithm can be used to find the multiplicative inverse (if it exists)
 - solving the problem:
 - Find $u, v | u^*m + v^*n = 1$
- Theorem: a number m has a multiplicative inverse $m^{-1} \text{ (mod } n)$ if and only if m and n are relatively prime (co-prime)
 - that is, if and only if $\gcd(m, n) = 1$



Extended Euclid's Algorithm

- Not only calculates the $d = \gcd(a, b)$, but also two integer x and y (of opposite sign) such that:

$$x a + y b = d = \gcd(a, b)$$

- From Euclid's Algorithm:

➤ $r_k = r_{k-2} \bmod r_{k-1} = r_{k-2} - q_k r_{k-1}$, where: $q_k = \lfloor r_{k-2} / r_{k-1} \rfloor$

with:

➤ $r_{-1} = a$

➤ $r_0 = b$

- Writing the previous equation as function of a and b :

➤ $r_1 = a \bmod b = a - q_1 b =^{(\text{def})} x_1 a + y_1 b$

➤ $r_2 = b \bmod r_1 = b - q_2(r_1) = b - q_2(x_1 a + y_1 b) = -q_2 x_1 a + (1 - q_2 y_1) b =^{(\text{def})} x_2 a + y_2 b$

➤ $r_3 = r_1 \bmod r_2 = r_1 - q_3 r_2 = (x_1 - q_3 x_2) a + (y_1 - q_3 y_2) b =^{(\text{def})} x_3 a + y_3 b$

➤ ...

➤ $r_k = r_{k-2} \bmod r_{k-1} =^{(\text{def})} x_k a + y_k b$

with:

➤ $x_k = x_{k-2} - q_k x_{k-1}$, with: $x_{-1}=1$, $x_0=0$

➤ $y_k = y_{k-2} - q_k y_{k-1}$, with: $y_{-1}=0$, $y_0=1$



Extended Euclid's Algorithm (cont.)

- Algorithm:

- **set**

- $r_{-1} = a$, $x_{-1}=1$, $y_{-1}=0$
 - $r_0 = b$, $x_0=0$, $y_0=1$

- **compute**

- $r_k = r_{k-2} \text{ mod } r_{k-1} = r_{k-2} - q_k r_{k-1}$, where $q_k = \lfloor r_{k-2} / r_{k-1} \rfloor$
 - $x_k = x_{k-2} - q_k x_{k-1}$
 - $y_k = y_{k-2} - q_k y_{k-1}$

- **until**

- $r_{n+1} = 0$

- **then, it is**

- $d = r_n = x_n a + y_n b$

- **that gives both the GCD d and the values x and y such that**

- $d = x a + y b$



Computation of the multiplicative inverse (modulo n)

- If the parameter a is a modulus n , and the parameter b is an integer m such that $GCD(m,n)=1$, then the algorithm gives the coefficients x and y such that

$$x*n + y*m = 1$$

$$y*m = 1 - xn$$

that can be written as

$$y*m = 1 + kn$$

that says that y is the multiplicative inverse of m modulo n

- Note
 - If the value of the coefficient y is negative ($-w$), the residue modulo n (between 0 and $n-1$) multiplicative inverse of m can be simply obtained as

$$y + n = n - w$$



Extended Euclid's Algorithm - Example 1

k	q_k	r_k	x_k	y_k	
-1		43	1	0	
0		35	0	1	
1	$43 = 1 \cdot 35 + 8$	8	1	-1	$8 = 1 \cdot 43 + (-1) \cdot 35$
2	$35 = 4 \cdot 8 + 3$	3	-4	5	$3 = (-4) \cdot 43 + 5 \cdot 35$
3	$8 = 2 \cdot 3 + 2$	2	9	-11	$2 = 9 \cdot 43 + (-11) \cdot 35$
4	$3 = 1 \cdot 2 + 1$	1	-13	16	$1 = (-13) \cdot 43 + 16 \cdot 35$
5	$2 = 2 \cdot 1 + 0$	0			

- Algorithm start with $k=1$
- At each step the new coefficients r_k , q_k are calculated:
 - $q_k = \lfloor r_{k-2} / r_{k-1} \rfloor$
 - $r_k = r_{k-2} \bmod r_{k-1}$
 - that is: $r_k = r_{k-2} - q_k r_{k-1}$
- From r_k , q_k , the new values of x_k e y_k are calculated:
 - $x_k = x_{k-2} - q_k x_{k-1}$
 - $y_k = y_{k-2} - q_k y_{k-1}$



Extended Euclid's Algorithm - Example 2

- Solve $1759x + 550y = \gcd(1759, 550)$

i	q_i	r_i	x_i	y_i
-1		1759	1	0
0		550	0	1
1	3	109	1	-3
2	5	5	-5	16
3	21	4	106	-339
4	1	1	-111	355

- Result:

- $d = 1; x = -111; y = 355$
- $(355)*550 + (-111)*1759 = 1$
 - $(355)*550 = 1 + k 1759$
 - 355 is the multiplicative inverse of 550 (mod 1759)



Galois Fields GF(p)

- If p is prime, all integers x with $x < p$ are relatively prime with p
- $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$ with arithmetic operations modulo prime p form a finite field (aka known as Galois Field)
 - since have multiplicative inverses
- GF(p) = Galois field of order p
- Arithmetic is “well-behaved” and can do addition, subtraction, multiplication, and division without leaving the field GF(p)



Example – Arithmetic in GF(7)

Addition modulo 7

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Multiplication modulo 7

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Additive and multiplicative
inverses modulo 7

w	$-w$	w^{-1}
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6



Euler Totient Function $\phi(n)$

- When doing arithmetic modulo n
- **Complete set of residues** is: $0 \dots n-1$
- **Reduced set of residues** is those numbers (residues) which are relatively prime to n
 - eg for $n=10$,
 - complete set of residues is $\{0,1,2,3,4,5,6,7,8,9\}$
 - reduced set of residues is $\{1,3,7,9\}$
- Number of elements in reduced set of residues is called the **Euler Totient Function $\phi(n)$**



Euler Totient Function $\phi(n)$

- To compute $\phi(n)$ need to count number of elements to be excluded
- In general need prime factorization, but
 - for p (p prime) $\phi(p) = p-1$
 - for $p \cdot q$ (p, q prime) $\phi(p \cdot q) = (p-1)(q-1)$
- Examples
 - $\phi(37) = 36$
 - $\phi(21) = (3-1) \times (7-1) = 2 \times 6 = 12$



Euler's Theorem

- $a^{\phi(n)} \bmod n = 1$
 - where $\gcd(a, n) = 1$
- e.g.
 - $a=3; n=10; \phi(10)=4;$
 - hence $3^4 = 81 = 1 \bmod 10$
 - $a=2; n=11; \phi(11)=10;$
 - hence $2^{10} = 1024 = 1 \bmod 11$
- It generalizes the Fermat's (Little) Theorem that says:
 - $a^{p-1} \bmod p = 1$
 - where p is prime and $\gcd(a, p) = 1$
- Corollary from Euler's Theorem
 - $a^{k\phi(n)+1} \bmod n = a$



Primitive Roots

- Consider the equation $a^m \bmod n = 1$
 - If $\gcd(a, n) = 1$ then m does exist
 - Euler's theorem gives $m = \phi(n)$, but may be smaller
 - note: once powers reach m , cycle will repeat: $a^{m+k} = a^m \cdot a^k = a^k$
- The smallest m such that $a^m = 1$ is called multiplicative order of a modulo n
- A number g is a primitive root modulo n if every number coprime to n is congruent to a power of g modulo n
 - g is also called "generator of the multiplicative group of integers modulo n " (the reduced set of residues)
 - $\forall b, \gcd(b, n) = 1, \exists k : g^k \equiv b \pmod{n}$
 - such k is called the index or discrete logarithm of b to the base g modulo n
 - if $n=p$ is prime, then successive powers of g "generate" the group mod p
 - if the multiplicative order of a modulo n is $m = \phi(n)$ then a is a primitive root (modulo n)



Discrete Logarithms

- The inverse problem to exponentiation is to find the **discrete logarithm** of a number modulo p
- That is to find x where $a^x \equiv b \pmod{p}$
- Written as $x = \log_a b \pmod{p} = d\log_{a,p}(b)$
- If a is a primitive root and p is prime then always exists, otherwise may not
 - **Examples**
 - $x = \log_3 5 \pmod{13}$ has no answer
 - $x = \log_2 5 \pmod{13} = 9$, by trying successive powers
 - **Note, in case of modulus n not prime, it exists if:**
 - a is primitive root, b is co-prime with n
- Whilst exponentiation is relatively easy, finding discrete logarithms is generally a **hard** problem



Primality Testing

- Often need to find large prime numbers
- Traditionally **check by** using **trial division**
 - i.e. divide by all numbers (primes) in turn less than the square root of the number
 - only works for small numbers
- Alternatively can use statistical primality tests based on properties of primes
 - for which all prime numbers satisfy property
 - but some composite numbers, called pseudo-primes, also satisfy the property



Miller Rabin Algorithm

- A test based on Fermat's Theorem
- Algorithm is:

TEST (n) is:

1. Find integers k, q , with $k > 0$, q odd, so that $(n-1) = 2^k q$
2. Select a random integer a , $1 < a < n-1$
3. if $a^q \text{ mod } n = 1$ then return ("maybe prime");
4. for $j = 0$ to $k - 1$ do
 5. if $(a^{2^j q} \text{ mod } n = n-1)$
then return("maybe prime")
6. return ("composite")



Miller Rabin Algorithm (cont.)

- if Miller-Rabin returns “composite” the number is definitely not prime
- Otherwise is a prime or a pseudo-prime
- Chance it detects a pseudo-prime is $< \frac{1}{4}$
- Hence if repeat test with different random a then chance n is prime after t tests is:
 - $\Pr(n \text{ prime after } t \text{ tests}) = 1 - 4^{-t}$
 - eg. for $t=10$ this probability is > 0.99999



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Public Key (asymmetric) Cryptography

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Public-Key Cryptography

- Also referred to as asymmetric cryptography or two-key cryptography
- Probably most significant advance in the 3000 year history of cryptography
 - **public invention due to Whitfield Diffie & Martin Hellman in 1975**
 - at least that's the first published record
 - known earlier in classified community (e.g. NSA?)
- Is asymmetric because
 - **who encrypts messages or verify signatures cannot decrypt messages or create signatures**
 - **more in general, operation performed by two parties use different key values**



Public-Key Cryptography (cont.)

- Public-Key cryptography uses clever application of number theoretic concepts and mathematical functions rather than permutations and substitutions
- Makes use of "trapdoor functions"
 - **a trapdoor function is a function $f(x)$ that is fast to be computed, while its inverse is hard to be computed, unless a secret information t (the trapdoor) is known**
 - when t is known, it is also easy to compute x from $f(x)$
 - **example of trapdoor function:**
 - prime number product
 - it is easy to find the product of two big prime numbers
 - it is hard to factorize a composite (two factors) big integer
 - it is easy to factorize a composite (two factors) big integer when one of the two factor is known



Public-Key vs. Secret Cryptography

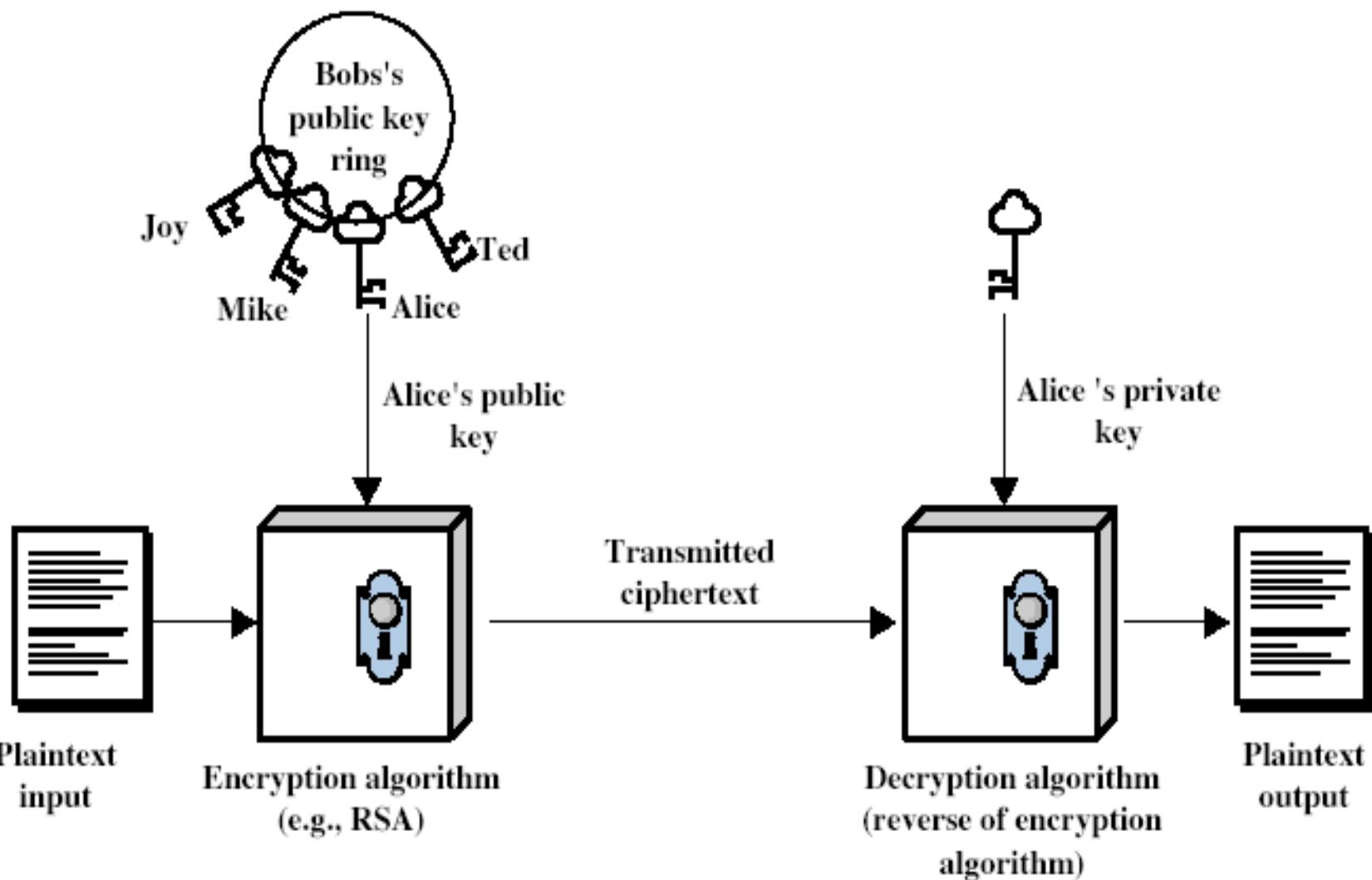
- All secret key algorithms do the same thing
 - **they take a block and encrypt it in a reversible way**
- All hash (and MAC) algorithms do the same thing
 - **they take a message and perform an irreversible transformation**
- Instead, public key algorithms look very different
 - **in how they perform their function**
 - **in what functions they perform**
- They have in common: a private and a public quantities associated with a principal



Public-Key vs. Secret Cryptography (cont.)

- Pub-key vs. secret key management
 - **With symmetric/secret-key cryptography**
 - you need a secure method of telling your partner the key
 - you need a separate key for everyone you might communicate with
 - **Instead, with public-key cryptography, keys do not have to be secretly shared**
 - **Public-key cryptography often uses two keys:**
 - a public-key, which may be known by anybody, and can be used to encrypt messages, or verify signatures
 - a private-key, known only to the recipient, used to decrypt messages, or sign (create) signatures
 - it is computationally easy to en/decrypt messages when key is known
 - it is computationally infeasible to find decryption key knowing only encryption key (and vice-versa)
 - **Some asymmetric algorithms don't use keys at all!**

Public-Key Cryptography





Public-Key vs. Secret Cryptography (cont.)

- Public key cryptography can do anything secret key cryptography can do, but..
 - **simpler key initialization**
 - **more accurate key-to-entity association**
 - private key is known only by the owner
 - **slower execution**
 - orders of magnitude slower than the best known secret key cryptographic algorithms
- They are usually used only for things secret key cryptography can't do (or can't do in a suitable way)
- Complements rather than replaces secret key crypto
 - **often it is mixed with secret key technology**
 - **e.g. public key cryptography might be used in the beginning of communication for authentication and to establish a temporary shared secret key used to encrypt the conversation**



Why Public-Key Cryptography?

- Can be used to:
 - key distribution – secure communications without having to trust a KDC with your key (key exchange)
 - digital signatures – verify a message is come intact from the claimed sender (authentication)
 - encryption/decryption - secrecy of the communication (confidentiality)
- Note:
 - public-key cryptography simplifies but not eliminates the problem of key management
 - some algorithms are suitable for all uses, others are specific
- Example of public key algorithms:
 - RSA, which does encryption and digital signature
 - DSS, which do digital signature but not encryption
 - Diffie-Hellman, which allows establishment of a shared secret
 - Fiat-Shamir identification scheme, which only do authentication



Security of Public Key Schemes

- Security of public-key algorithms still relies on key size (as for secret-key algorithms)
- Like private key schemes brute force exhaustive search attack is always theoretically possible
 - **But keys used are much larger (>512bits)**
- A crucial feature is that the private key is difficult to determine from the public key
 - **security relies on a large enough difference in difficulty between easy (en/decrypt) and hard (cryptanalyse) problems**
 - **often the hard problem is known, its just made too hard to do in practise**
 - requires the use of very large numbers
 - hence is slow compared to private key schemes

RSA Algorithm



Rivest, Shamir, and Adleman (RSA)

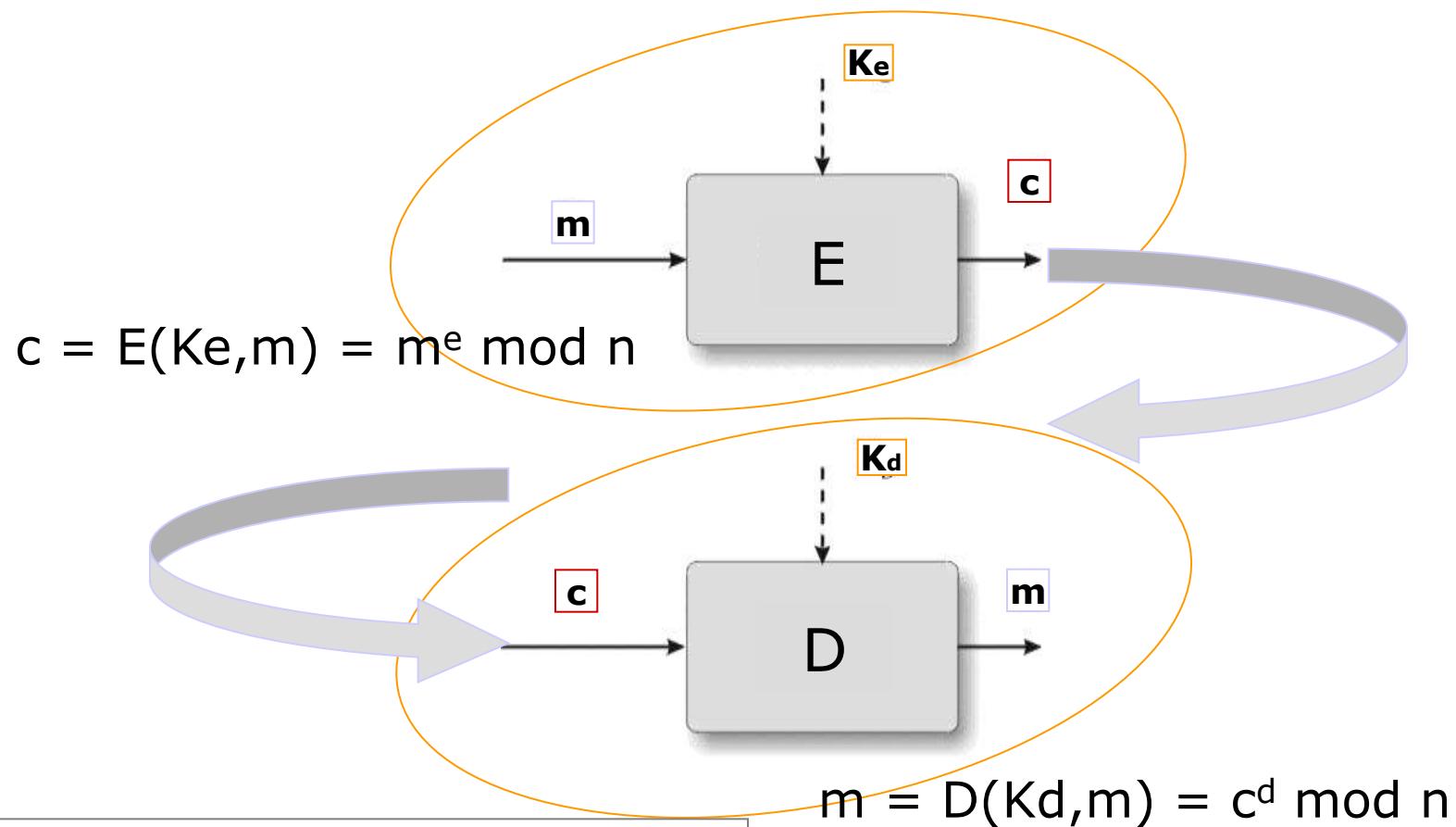
- By Rivest, Shamir & Adleman of MIT in 1977
- Best known & widely used public-key scheme
- Based on exponentiation in a finite (Galois) field over integers modulo n
 - n.b. exponentiation takes $O((\log n)^3)$ operations (easy)
- uses large integers (eg. 1024 bits)
- security due to cost of factoring large numbers
 - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)
- The key length is variable
 - long keys for enhanced security, or a short keys for efficiency
- The plaintext block size (the chunk to be encrypted) is also variable
 - The plaintext block size must be smaller than the key length
 - The ciphertext block will be the length of the key
- RSA is much slower to compute than popular secret key algorithms like DES, IDEA, and AES



RSA Algorithm

- First, you need to generate a public key and a corresponding private key:
 - choose two large primes p and q (around 512 bits each or more)
 - p and q will remain secret
 - multiply them together (result is 1024 bits), and call the result n
 - it's practically impossible to factor numbers that large for obtaining p and q
 - compute $\phi(n) = (p-1)(q-1)$
 - choose a number e that is relatively prime (that is, it does not share any common factors other than 1) to $\phi(n)$
 - find the number d that is the multiplicative inverse of e mod $\phi(n)$
 - your public key is $KU = K^+ = \langle e, n \rangle$
 - your private key is $KR = K^- = \langle d, n \rangle$ (or $\langle d, p, q \rangle$)
- To encrypt a message m ($< n$), someone can use your public key
 - $c = m^e \text{ mod } n$
- Only you will be able to decrypt c , using your private key
 - $m = c^d \text{ mod } n$

Textbook RSA



- m **plaintext**
- c **ciphertext**
- K_e **encryption key (e.g. public key, K^U or K⁺)**
- K_d **decryption key (e.g. private key, K^R or K⁻)**



RSA Key Setup

- Each user generates a public/private key pair by:
 - **selecting two large primes at random p, q**
 - **computing their system modulus $n = p \cdot q$**
 - note $\phi(n) = (p-1)(q-1)$
 - **selecting at random the encryption key e**
 - where $1 < e < \phi(n)$, $\gcd(e, \phi(n)) = 1$
 - **solve following equation to find decryption key d**
 - $e \cdot d \equiv 1 \pmod{\phi(n)}$ and $0 \leq d \leq n$
- Publish their public encryption key: $KU = \{e, n\}$
- Keep secret private decryption key: $KR = \{d, n\}$ or $\{d, p, q\}$



RSA Use

- To encrypt a message m the sender:
 - obtains public key of recipient $KU=\langle e, n \rangle$
 - computes: $c = m^e \text{ mod } n$, where $0 \leq m < n$
- To decrypt the ciphertext c the owner:
 - uses their private key $KR=\langle d, n \rangle$
 - computes: $m = c^d \text{ mod } n$
- Note that the message m must be smaller than the modulus n
 - it is a block cipher, where block size depends on the length of the modulus
 - if m is longer, CBC or other block cipher encryption modes can be used
 - much slower than symmetric encryption



Why RSA Works

- Because of Euler's Theorem:

➤ $a^{\phi(n)} \bmod n = 1$

- where $\gcd(a,n)=1$

also:

➤ $a^{k\phi(n)} \bmod n = 1^k = 1$

and:

➤ $a^{k\phi(n)+1} \bmod n = a$

- In RSA have:

➤ $n=p \cdot q$

➤ $\phi(n)=(p-1)(q-1)$

➤ **carefully chosen e and d to be inverses mod $\phi(n)$**

- hence $e \cdot d = 1 + k \cdot \phi(n)$ for some k

- Encryption:

➤ $c = E(K_e, m) = m^e \bmod n$

- Decryption:

➤ $D(K_d, c) = c^d \bmod n = (m^e)^d \bmod n = m^{ed} \bmod n = m^{1+k\phi(n)} \bmod n = m$



RSA Example

RSA setup

- select primes: $p=17$ & $q=11$
- compute $n = pq = 17 \times 11 = 187$
- compute $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
- select e : $\gcd(e, 160) = 1$; choose $e=7$
- determine d : $de \equiv 1 \pmod{160}$ and $d < 160$ Value is $d=23$ since $23 \times 7 = 161 = 160 + 1$
- publish public key $K^+ = K^+ = \{7, 187\}$
- keep secret private key $K^- = K^- = \{23, 187\} = \{23, 17, 11\}$



RSA Example (cont)

Textbook RSA encryption/decryption:

- given message $M = 88$ (nb. $88 < 187$)

- encryption:

$$C = 88^7 \bmod 187 = 11$$

- decryption:

$$M = 11^{23} \bmod 187 = 88$$



Textbook RSA is not secure

- Textbook RSA encryption:
 - **public key:** (n, e) **Encrypt:** $c = m^e \pmod{n}$, with $m < n$
 - **private key:** (n, d) **Decrypt:** $m = c^d \pmod{n}$
- If $n = pq$ is large, the factorization of n is practically impossibly
- However, many attacks exist to Textbook RSA
 - **examples**
 - if e is small, and the message m is $< n^{1/e}$, then m^e is $< n$ and computation of c doesn't involve any modular reduction
 - if m is small, knowing c and $\{e, n\}$ a brute force search is possible
 - an improvement of this attack also exists, that does not require a brute force search
 - common modulus attack - if several keys share the same modulus n , if the same m is encrypted with two keys $\{e_1, n\}$ and $\{e_2, n\}$ with $\gcd(e_1, e_2) = 1$, an adversary who sees c_1 and c_2 and knows the two public keys can recover m
 - $\gcd(e_1, e_2) = 1 \rightarrow ue_1 + ve_2 = 1 \rightarrow c_1^u * c_2^v = m^{e_1 u} * m^{e_2 v} = m^{ue_1 + ve_2} = m$



Padded RSA

- $c = E(k_e, m) = (\text{pad}(m))^e \bmod n$
- $m = \text{pad}^{-1}(m')$, with $m' = c^d \bmod n$
- Different padding schemes exist, examples:
 - **PKCS#1 - v1.5**
 - **PKCS#1 - v2 OAEP (Optimal Asymmetric Encryption Padding)**



PKCS#1 - V1.5

- Encoded message $EM = 0x00 \parallel 0x02 \parallel PS \parallel 0x00 \parallel M$
where:
 - len(m) up to $L-11$ octets, where L is the octet length of the RSA modulus
 - **PS** is padding string consisting of pseudo-randomly generated nonzero octets
 - len(PS) at least eight octets



PKCS#1 - V2 OAEP

- Given:
 - **MGF(seed,len)**
 - Mask Generation Function that generates *len* pseudo-random octects using a given *seed*
 - **H(x)**
 - hash function with length *h*
- Encoded message $EM = \text{maskedSeed} \parallel \text{maskedDB}$
 - **with $\text{len}(M) < \text{len}(EM) - 2h - 1$**
 - **Where:**
 - $P = \text{a parameter string or label or null string}$
 - data block $DB = H(P) \parallel PS \parallel 01 \parallel M$
 - with $\text{len}(DB) = h + \text{len}(PS) + 1 + \text{len}(M) = \text{len}(EM) - h$
 - $\text{seed} = \text{random octet string}$
 - with $\text{len}(\text{seed}) = h$
 - $\text{maskedDB} = DB \text{ XOR } \text{MGF}(\text{seed}, \text{len}(EM)-h)$
 - with $\text{len}(\text{maskedDB}) = \text{len}(EM)-h$
 - $\text{maskedSeed} = \text{seed} \text{ XOR } \text{MGF}(\text{maskedDB}, h)$
 - with $\text{len}(\text{maskedSeed}) = h$



RSA Security

- Three main approaches to attacking RSA:
 - **brute force key search**
 - brute force search on key space
 - infeasible given the key size
 - **cryptographic attacks**
 - try to find the private key by finding $\phi(n)$, by factoring modulus n and find p and q
 - infeasible given the size of n
 - other attacks in case p, q, e, d values are not selected properly
 - **timing attacks**
 - by measuring the time spent on running decryption



RSA Security (cont.)

- Cryptographic attacks if p, q, e, d are not selected properly:
 - **Modulus too small**
 - if the RSA key is too short, the modulus can be factored by just using brute force
 - **Low private exponent**
 - the smaller d is, the faster this operation goes
 - note: If the private exponent is small, the public exponent is necessarily large, so a public key with a large public exponent, is a good hint for attackers
 - **Low public exponent**
 - having a low public exponent makes the system vulnerable to certain attacks if used incorrectly
 - **Generator p and q close together**
 - if $p \approx q$, then $n \approx p^2$ and n can be efficiently factored using Fermat's factorization method



Progress in factorization (from Wikipedia)

RSA number	Decimal digits	Binary digits	Cash prize offered	Factored on	Factored by
RSA-100	100	330	US\$1000	April 1, 1991	Arjen K. Lenstra
RSA-110	110	364	US\$4429	April 14, 1992	Arjen K. Lenstra and M.S. Manasse
RSA-120	120	397	US\$5898	July 9, 1993	T. Denny et al.
RSA-129	129	426	US\$100	April 26, 1994	Arjen K. Lenstra et al.
RSA-130	130	430	US\$14527	April 10, 1996	Arjen K. Lenstra et al.
RSA-140	140	463	US\$17226	February 2, 1999	Herman te Riele et al.
RSA-150	150	496		April 16, 2004	Kazumaro Aoki et al.
RSA-155	155	512	US\$9383	August 22, 1999	Herman te Riele et al.
RSA-160	160	530		April 1, 2003	Jens Franke et al., University of Bonn
RSA-170	170	563		December 29, 2009	D. Bonenberger and M. Krone
RSA-576	174	576	US\$10000	December 3, 2003	Jens Franke et al., University of Bonn
RSA-180	180	596		May 8, 2010	S. A. Daniilov and I. A. Popovyan, Moscow State University
RSA-190	190	629		November 8, 2010	A. Timofeev and I. A. Popovyan
RSA-640	193	640	US\$20000	November 2, 2005	Jens Franke et al., University of Bonn
RSA-200	200	663		May 9, 2005	Jens Franke et al., University of Bonn
RSA-210	210	696		September 26, 2013	Ryan Propper
RSA-704	212	704	US\$30000	July 2, 2012	Shi Bai, Emmanuel Thomé and Paul Zimmermann
RSA-220	220	729		May 13, 2016	S. Bai, P. Gaudry, A. Kruppa, E. Thomé and P. Zimmermann
RSA-230	230	762		August 15, 2018	Samuel S. Gross, Noblis, Inc.
RSA-232	232	768		February 17, 2020	N. L. Zamarashkin, D. A. Zheltkov and S. A. Matveev.
RSA-768	232	768	US\$50000	December 12, 2009	Thorsten Kleinjung et al.
RSA-240	240	795		Dec 2, 2019	F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P.
RSA-250	250	829		Feb 28, 2020	F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P.

2007
End of challenge



Using both symmetric and asymmetric cryptography

- Confidentiality can be provided through either symmetric or asymmetric encryption
- Requires that the two (or more) parties share:
 - **the secret key, in case of symmetric encryption, or**
 - **the public key of the sender, in case of asymmetric encryption**
- Usually symmetric encryption is preferred when
 - **a long message have to be encrypted**
 - **multiple messages have to be sent**
- In this case, if the two parties share only public keys, public key cryptography can be used for exchanging a symmetric (secret) key
 - **this secret key is sometimes referred as session key**

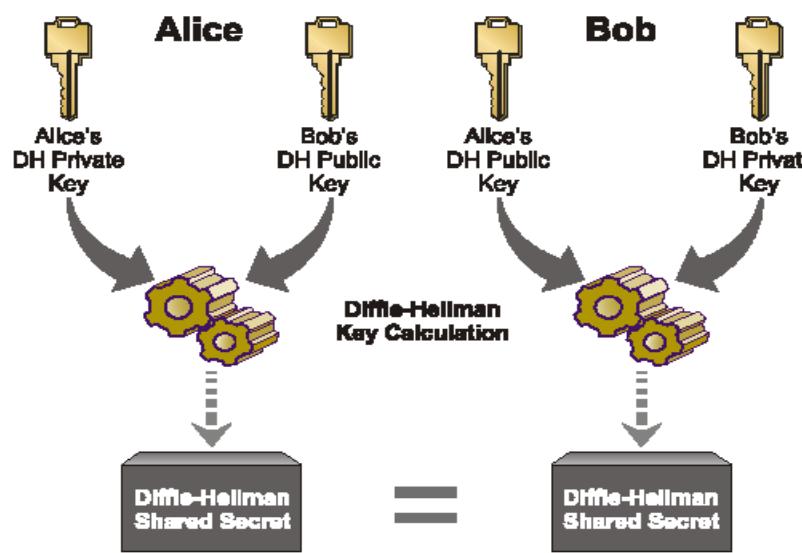


Using both symmetric and asymmetric cryptography (cont.)

- Example of encryption of a message m from A to B using symmetric cipher without having a pre-shared symmetric secret key, in one pass:
 - $A \rightarrow B: E(K_s, m), \{K_s\}K_B^+$
 - where:
 - $E(K, x)$: symmetric encryption (e.g. AES-CBC)
 - $\{x\}K^+$: public key encryption (e.g. RSA)
- Other mechanisms are possible, involving more passes (exchanges) using a key establishment (key agreement) protocol
 - e.g. using authenticated Diffie-Hellman exchange

Diffie-Hellman (DH)

Diffie-Hellman





Diffie-Hellman

- First public-key type scheme proposed
- By Diffie & Hellman in 1976 along with the exposition of public key concepts
 - **now know that James Ellis (UK CESG) secretly proposed the concept in 1970**
 - predates RSA
 - **less general than RSA: it does neither encryption nor signature**
- Is a practical method for public exchange of a secret key
 - **allows two individuals to agree on a shared secret (key)**
 - **It is actually used for key establishment**
- Used in a number of commercial products

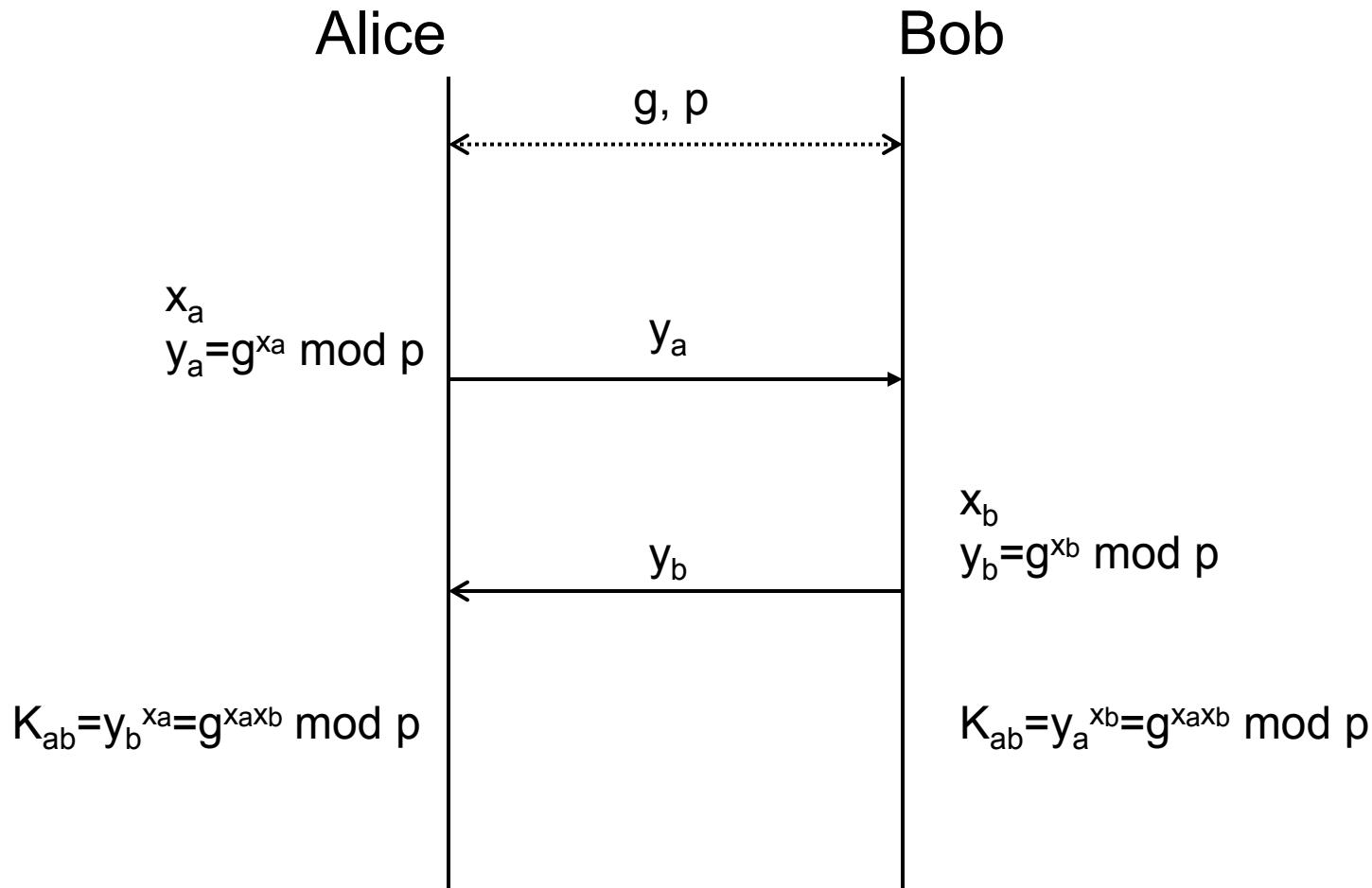


Diffie-Hellman Setup

Diffie-Hellman setup:

- all users agree on global parameters:
 - **$p = \text{a large prime integer or polynomial}$**
 - **$g = \text{a primitive root mod } p$**
- each user (eg. A) generates their key
 - **chooses a secret key (number): $x_A < p$**
 - **compute their public key: $y_A = g^{x_A} \text{ mod } p$**
- each user makes public that key y_A

Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange

Key exchange:

- Shared key K_{AB} for users A & B can be computed as:

$$\begin{aligned} K_{AB} &= g^{x_A x_B} \bmod p \\ &= y_B^{x_A} \bmod p \quad (\text{which A can compute}) \\ &= y_A^{x_B} \bmod p \quad (\text{which B can compute}) \end{aligned}$$

- K_{AB} can be used as session key in secret-key encryption scheme between A and B
- Attacker must solve discrete log
 - **hard problem if p is chosen properly**
- Requires an integrity protected channel
 - **otherwise it is vulnerable to Man-In-The-Middle (MITM) attack**



Diffie-Hellman - Example

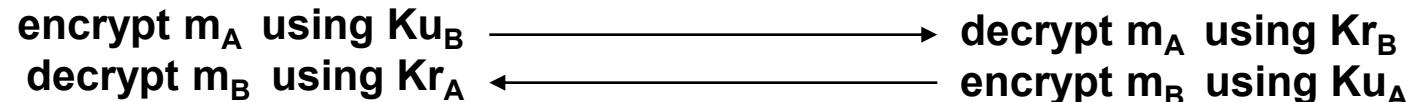
- users Alice & Bob who wish to swap keys:
- agree on prime $p=353$ and $g=3$
- select random secret keys:
 - A chooses $x_A=97$, B chooses $x_B=233$
- compute public keys:
 - $y_A = 3^{97} \text{ mod } 353 = 40$ (Alice)
 - $y_B = 3^{233} \text{ mod } 353 = 248$ (Bob)
- compute shared session key as:
 - $K_{AB} = y_B^{x_A} \text{ mod } 353 = 248^{97} = 160$ (Alice)
 - $K_{AB} = y_A^{x_B} \text{ mod } 353 = 40^{233} = 160$ (Bob)

Security uses of public key cryptography

- Transmitting over an insecure channel

 - e.g. RSA

 - each party has a <public key, private key> pair (Ku, Kr)
 - each party encrypts with the public key of the other party



- Secure storage on insecure media

 - e.g. RSA

 - encrypt with public key, decrypt with private key
 - useful when you can let third party to encrypt data

- Data authentication (Digital signature)

 - e.g. DSA, RSA signature

- Key establishment

 - e.g. Diffie-Hellman

Security uses of public key cryptography

- Peer Authentication (identification)

- **Zero Knowledge Proof schemes**

- prove that you know a secret without leaking any information
 - an entity A (prover) identifies itself by proving knowledge of a secret to any verifier B, without revealing any information about the secret, not known or computable by B prior to execution of the algorithm

- **RSA**

- authentication by proving the knowledge of the private key
- encrypt r using Ku_B** → **decrypt to r using Kr_B**
- ↔ **r**

- Note

- **Public key cryptography has specific algorithm for specific function such as**

- data encryption
 - MAC/digital signature
 - key establishment
 - peer authentication



Pros and cons of Public key cryptography

● Pros

- **Every users have to keep only one secret (the private key)**
 - public keys of other users can be verified through a trusted third party infrastructure (e.g. PKI)
- **The total number of keys for N users is $2N$**
 - instead, with symmetric cryptography $n(n-1)/2$ keys are needed

● Cons

- **Slower**
 - known public-key cryptographic algorithms are orders of magnitude slower than the best known secret key cryptographic algorithms



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Authenticity: Message Authentication and Digital Signature

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Message Authentication

- Message authentication is concerned with:
 - **protecting the integrity of a message**
 - **origin authentication**
 - validating identity of originator
 - **in some cases, also non-repudiation of origin (dispute resolution)**

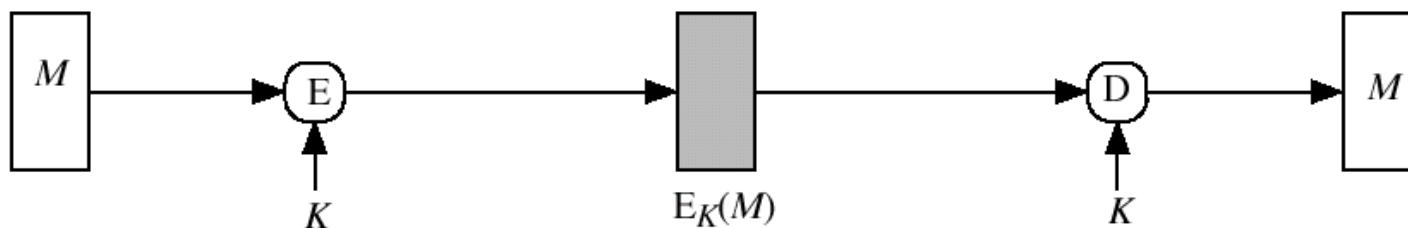
- Possible approaches:
 - **Symmetric mechanisms**
 - Symmetric encryption
 - sometimes together with an internal integrity check
 - Message Authentication Code
 - keyed one-way functions
 - **Asymmetric mechanisms**
 - Asymmetric encryption
 - Digital signature

Message Authentication using symmetric keys

Msg. Auth. - Secret-key Encryption

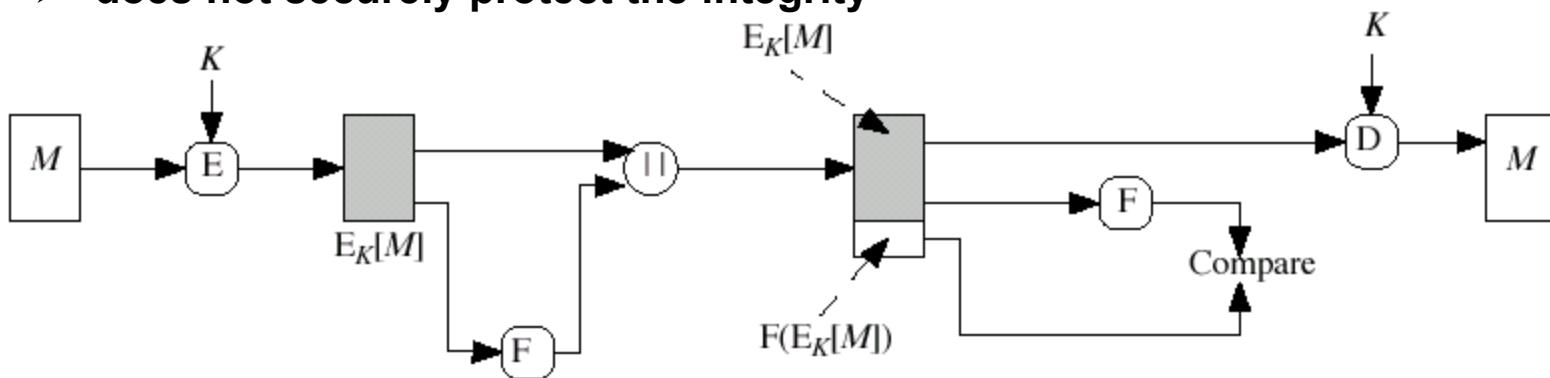
- Symmetric encryption:

- **encryption may provide both confidentiality and origin authentication**
- **however, need to recognize corrupted messages**
 - based on the received message or with an explicit internal integrity check (see next slide)

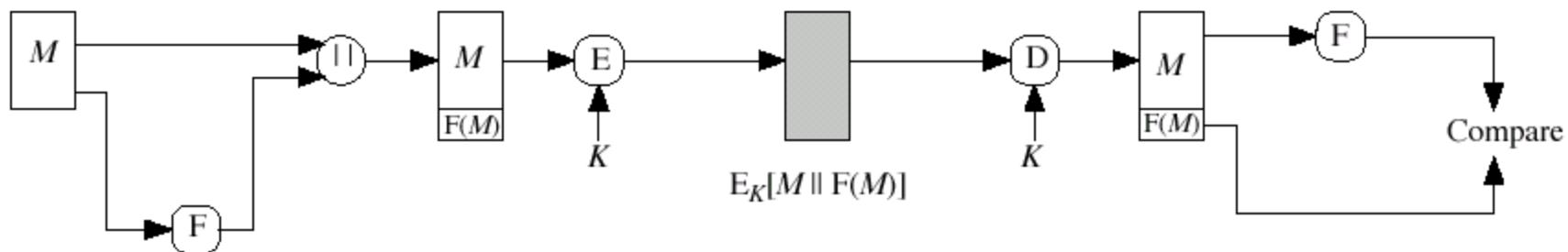


Msg. Auth. - Secret-key Encryption (cont.)

- External error control (checksum):
 - does not securely protect the integrity



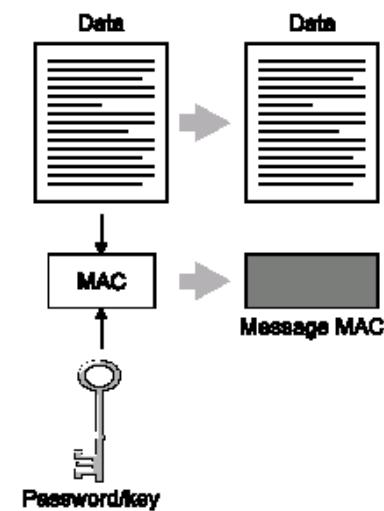
- Internal integrity check, through:
 - a manipulation detection code (a sort of robust checksum)





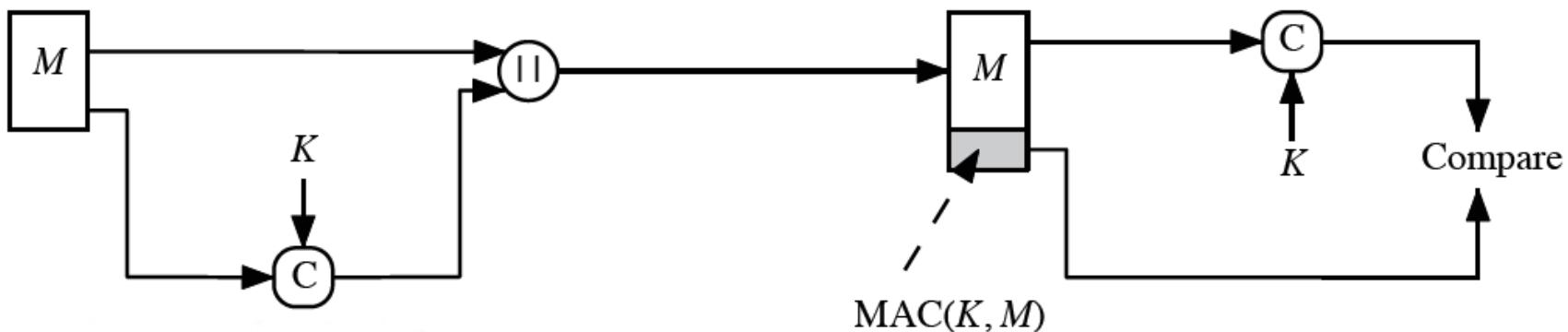
Message Authentication Code (MAC)

- Cryptographic checksum, generated by an algorithm that creates a small fixed-sized block
 - depending on both message and a secret key K
 - $\text{MAC} = C_K(M) = C(K, M)$
 - condenses a variable-length message M to a fixed-sized authenticator
 - it doesn't need to be reversible
 - is a many-to-one function
 - potentially many messages have same MAC
 - but finding these needs to be very difficult



Message Authentication Code (MAC)

- Use of MAC for message authentication:
 - appended to message as a signature
 - receiver performs same computation on message and checks it matches the MAC
 - provides assurance that message is unaltered and comes from sender
 - can be used also without enforcing confidentiality





Message Authentication Code (MAC) (cont.)

- In case secrecy is also required
 - **use of encryption with separate key**
 - **can compute MAC either before or after encryption**
 - is generally regarded as better done before
- Why use a MAC?
 - **sometimes only authentication is needed**
 - **sometimes need authentication to persist longer than the encryption (eg. archival use)**
- MAC is similar but not equal to digital signature



Requirements for a MAC function

- MAC functions have to satisfy the following requirements:
 - knowing a message and MAC, is infeasible to find another message with same MAC
 - is infeasible to find two messages with same MAC
- Additional properties:
 - MAC value should be uniformly distributed
 - MAC should depend equally on all bits of the message
- Properties similar to hash functions
 - in addition, MAC uses an input key

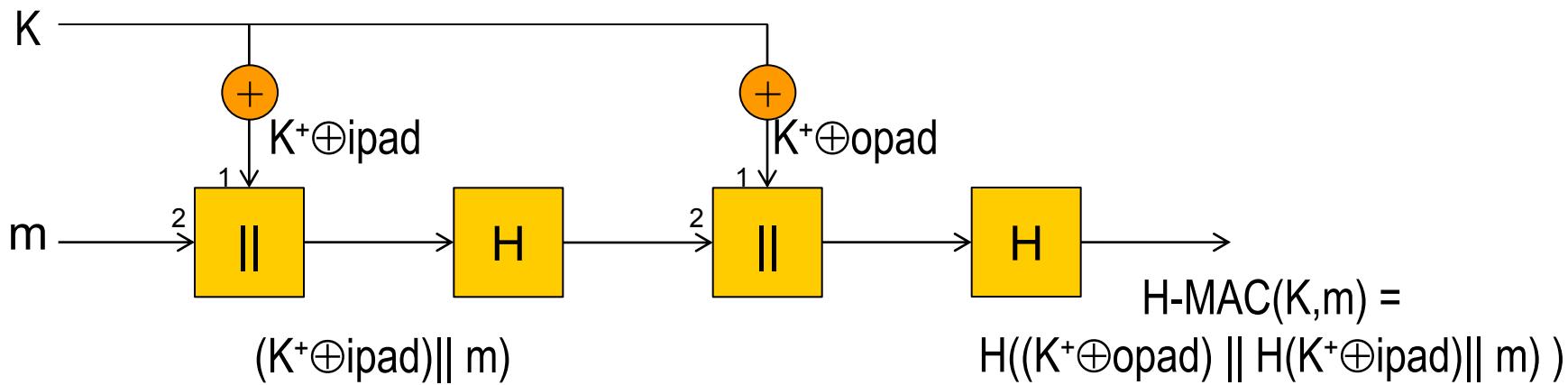


Hash Message Authentication Code (H-MAC)

- Mechanism for message authentication using cryptographic hash functions in combination with a secret shared key
- Specified as Internet standard RFC2104
- HMAC can be used with any iterative cryptographic hash function, e.g., MD5, SHA-1, SHA-256, etc
 - **the cryptographic strength of H-MAC depends on the properties of the underlying hash function**

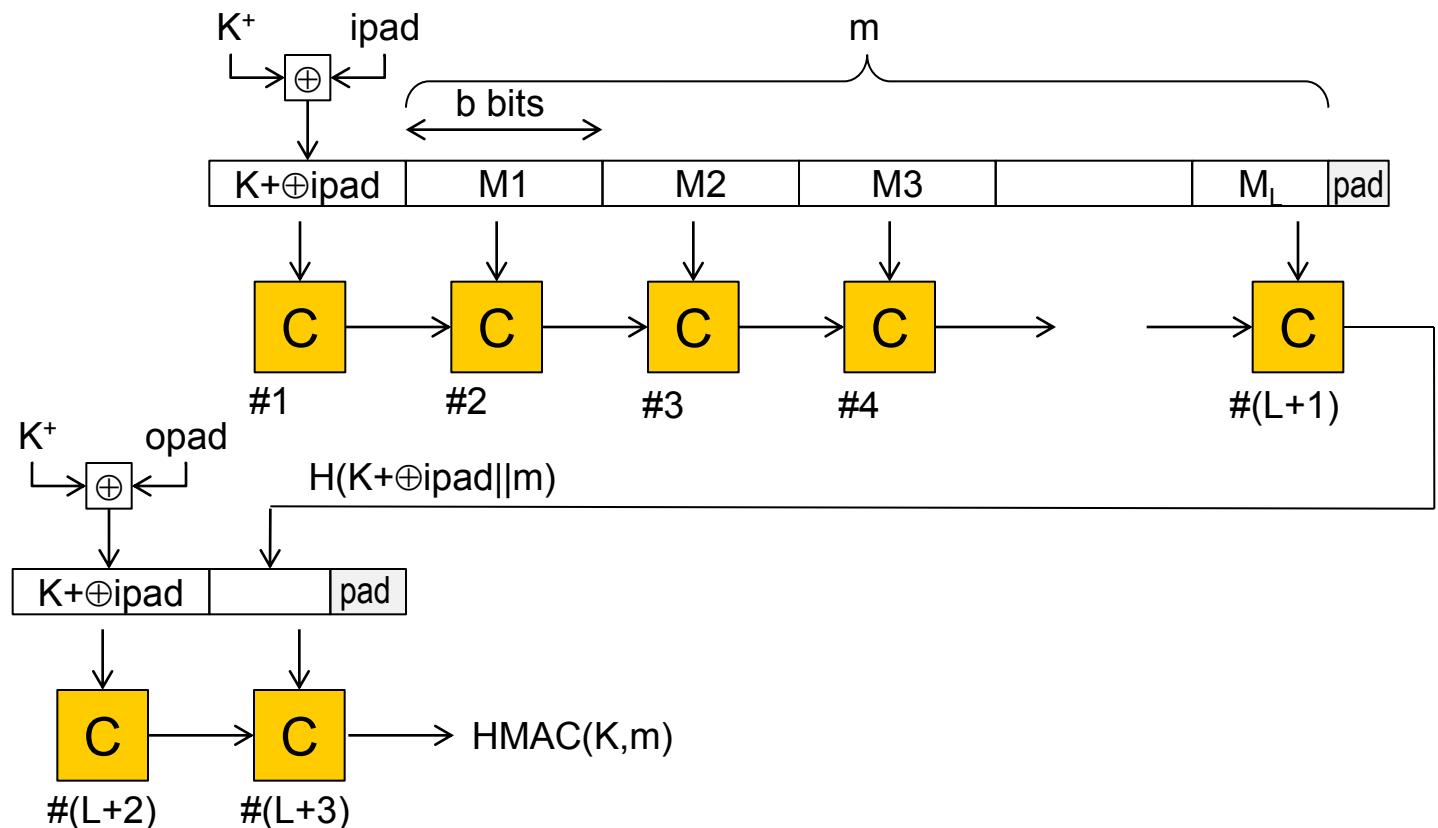
H-MAC (cont.)

- $\text{HMAC}(K, m) = \text{Hash}[(K^+ \text{ XOR opad}) \parallel \text{Hash}[(K^+ \text{ XOR ipad}) \parallel m]]$
 - where **K+** is the key 0-padded out to size b
 - b is the size of the processing block
 - e.g. b = 512bits = 64bytes for SHA1
 - if K is longer than b bytes it is first hashed using H
 - and opad, ipad are specified padding constants
 - ipad = the byte 0x36 repeated b/8 times
 - opad = the byte 0x5C repeated b/8 times



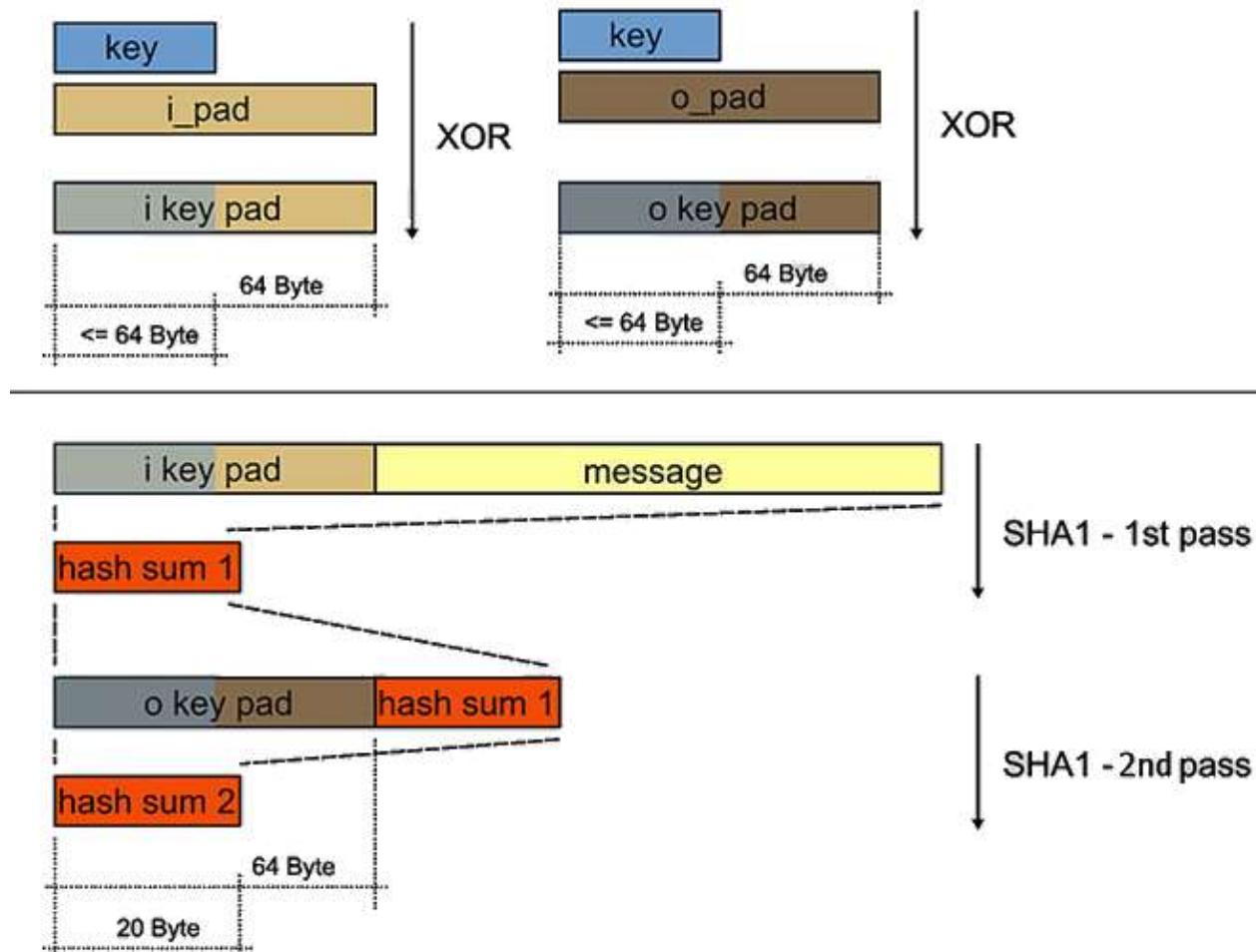
H-MAC (cont.)

- Overhead is just 3 more hash inner calculations (compression function C) than the message needs alone for computing message digest





Example - H-MAC-SHA1





Truncated H-MAC

- A well-known practice with MACs is to truncate the output of the MAC and output only part of the bits
 - **Advantage: less information on the hash result available to an attacker**
 - **Disadvantage: less bits to predict for the attacker**
- It is recommended to let the output length t be not less than half the length of the hash output and not less than 80 bits
- HMAC that uses a hash function H with t bits of output can be denoted as $\text{HMAC-}H\text{-}t$
 - **Example, HMAC-SHA1-80 denotes HMAC computed using the SHA-1 function and with the output truncated to 80 bits**

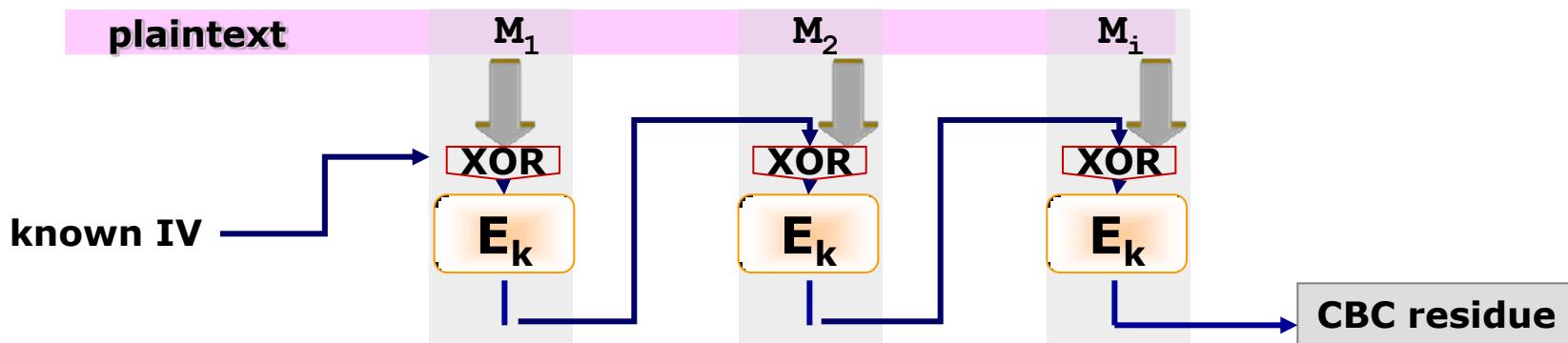


MAC Security

- Attacks:
 - Cryptanalytic attacks
 - Brute-force attacks
- Transient effect
 - a published breaking of a MAC scheme would lead to the replacement of that scheme, but would have no adversarial effect on information authenticated in the past
 - this is in contrast with encryption, where information encrypted today may suffer from exposure in the future if, and when, the encryption algorithm is broken

Using Symmetric Ciphers for MAC

- If a cryptographic algorithm is available with CBC mode, a way of generating a MAC is to compute the CBC but keep only the last block (named CBC residue) as MAC value



- CBC-MAC standard mode use $IV=0$
- E.g. Data Authentication Algorithm (DAA) (now obsoleted) is a CBC-MAC based on DES
- Can use also other block cipher chaining modes and use final block



Using Symmetric Ciphers for MAC (cont.)

- Another approach could be to encrypt the hash
 - $\text{MAC}_K(m) \equiv E_K(H(m))$

- The main drawbacks of MAC based on symmetric ciphers are:
 - **the lower speed (e.g. compared with HMAC)**
 - **the size of the output that may be too small for security (it depends on the block cipher)**

Authenticated Encryption



Authenticated Encryption

- Sometimes both message authentication and encryption are required
- Authenticated encryption (AE) is a cryptographic system that simultaneously protects confidentiality and authenticity (integrity)
- There are four common approaches to providing both confidentiality and authenticity for a message m :
 - **Hash-then-Encrypt ***
 - $E_K(m||H(m))$
 - **MAC-then-Encrypt**
 - $E_{K2}(m|| MAC_{K1}(m))$
 - **Encrypt-then-MAC**
 - $C|| MAC_{K1}(C)$, where $C=E_{K2}(m)$
 - **Encrypt-and-MAC**
 - $E_{K2}(m) || MAC_{K1}(m)$
- Methods 2, 3, and 4 use two different keys



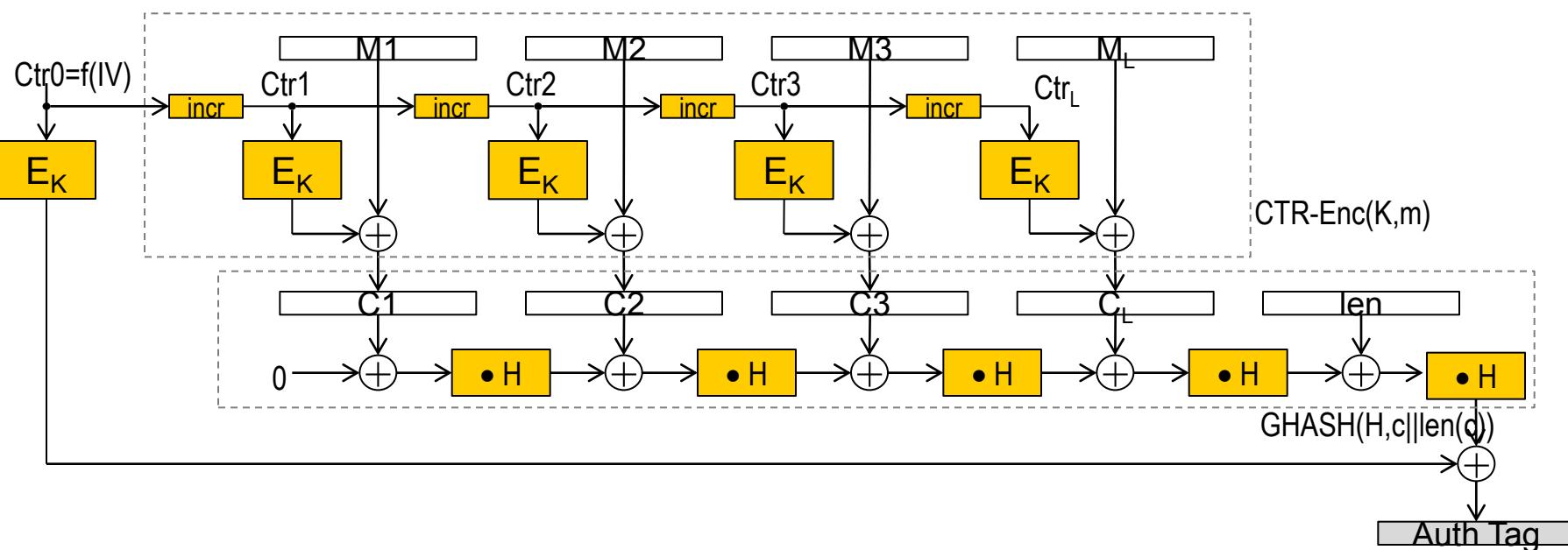
Authenticated Encryption (cont.)

- Example:
 - **CCM (Counter mode with CBC-MAC)**
 - Encrypt-and-MAC
 - ciphertext: $c = \text{CTR-Enc}(K, m)$
 - auth tag: $t = \text{Enc}(K, Ctr0) \oplus \text{CBC-MAC}(K, N||m)$
 - » where N is a nonce value, $Ctr0$ is the first generated counter (then $Ctr1, Ctr2, \dots$ are used for encrypting)
- A more efficient design of an AE system is to process the message only one time (just one pass)
 - instead of separately encrypting and computing the MAC
 - e.g. Galois/Counter Mode (GCM)

Example: Galois/Counter Mode (GCM)

- It is Encrypt-and-MAC:

- ciphertext: $c = \text{CTR-Enc}(K, m)$, where $\text{Ctr0} = f(\text{IV})$
- auth tag: $t = E(K, \text{Ctr0}) \oplus \text{GHASH}(H, c \parallel \text{len}(c))$
 - where $\text{GHASH}(k, x)$ is a non-cryptographic keyed hash function
 - $H = \text{Enc}(K, 0)$
 - $X \cdot Y$ is multiplication operation for the binary Galois field of 2^{128} elements
 - modular multiplication of binary polynomials of degree less than 128





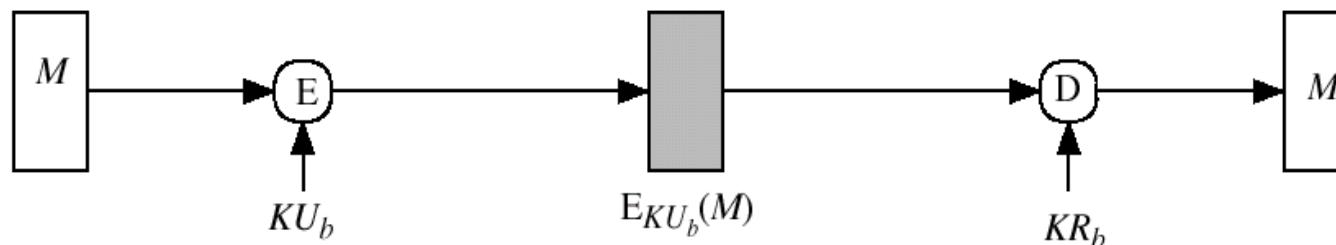
Authenticated Encryption with Associated Data

- Authenticated Encryption with Associated Data (AEAD)
 - in addition to the plaintext ' m ' (that has to be AEed) there is extra data ' a ' that has to be only authenticated
 - e.g. for protecting a network packet:
 - packet header (A) only authenticated since it must be readable,
 - packet payload (m) authenticated and encrypted
- Examples:
 - **CCM (Counter mode with CBC-MAC) AEAD**
 - $c = \text{CTR-Enc}(K, m)$
 - $t = \text{CBC-MAC}(K, N || A || m) \oplus \text{Enc}(K, \text{Ctr}0)$
 - **GCM (Galois/Counter Mode) AEAD**
 - $c = \text{CTR-Enc}(K, m)$, where $\text{Ctr}0 = f(\text{IV})$
 - $t = \text{Enc}(K, \text{Ctr}0) \oplus \text{GASH}(H, A || c || \text{len}(A) || \text{len}(c))$, with $H = \text{Enc}(K, 0)$

Digital Signature

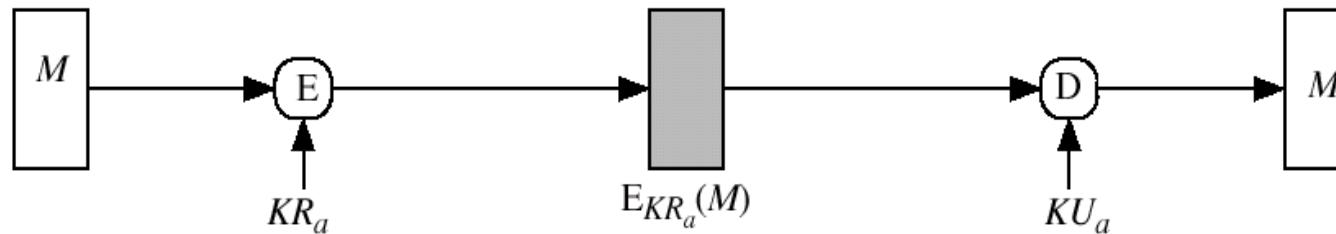
Msg. Auth. - Asymmetric Encryption

- Asymmetric encryption with public key: confidentiality

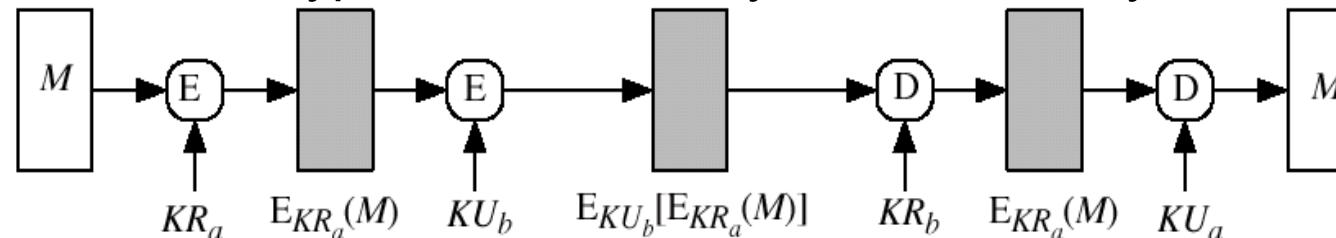


- Asymmetric encryption with private key: authentication

➤ however need to recognize corrupted messages



- Asymmetric encryption with both keys: confidentiality + authentication





Msg. Auth. - Asymmetric Encryption (cont.)

- if public-key encryption is used
 - **encryption with public key provides no proof of sender (no sender authentication)**
 - since anyone potentially knows public-key
 - **encryption with private key provides authentication of the sender**
 - if it is possible to distinguish, after decryption, between a valid message and a random string of bits
 - **both secrecy and authentication if**
 - sender “signs” message using their private-key
 - then encrypts with recipients public key
- Problems
 - need to recognize corrupted messages
 - the signs has the same cost of public-key encryption of the entire message



Digital Signature

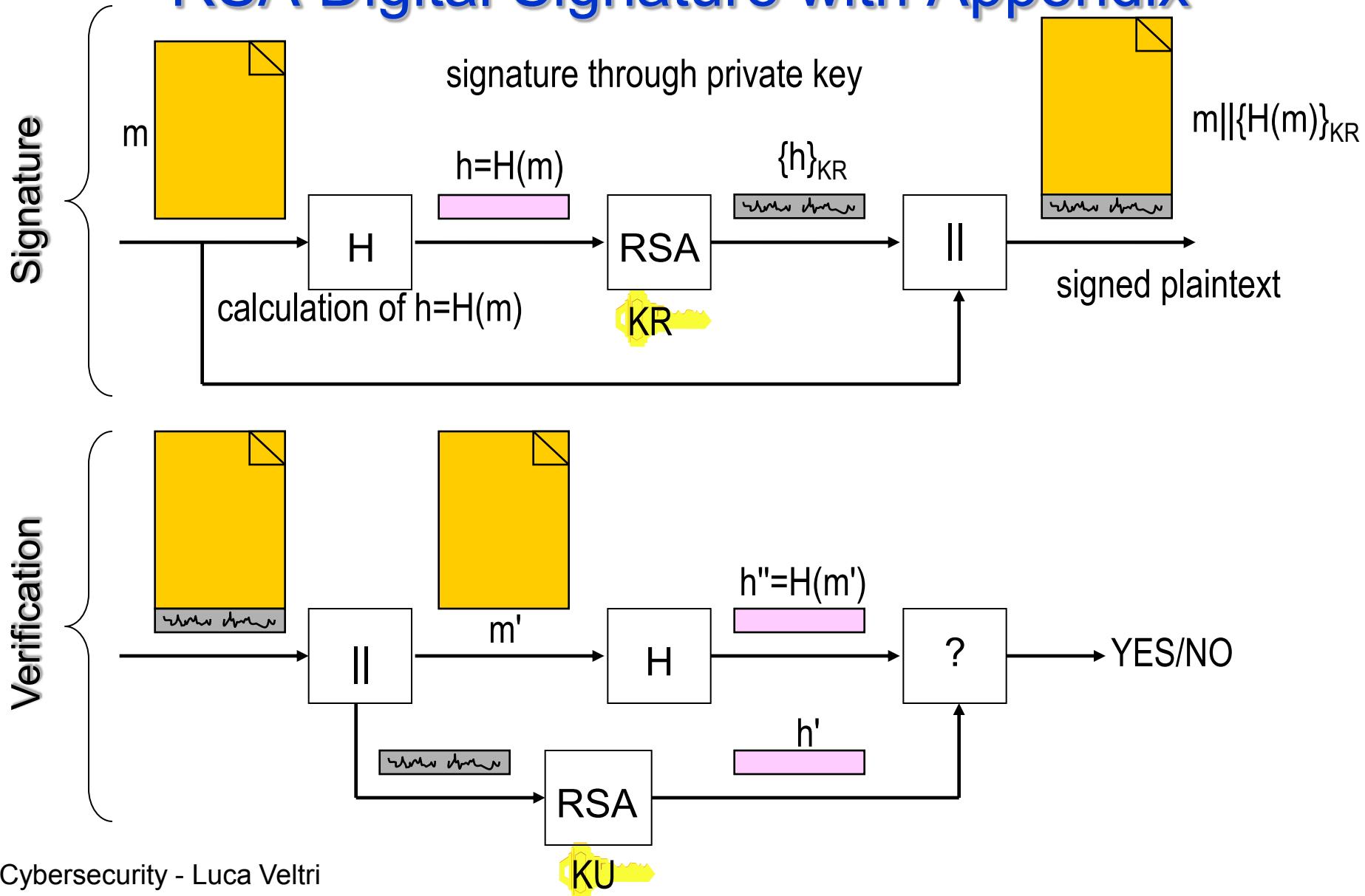
- Digital Signature is an application in which
 - a signer, say "Alice," "signs" a message m in such a way that
 - anyone can "verify" that the message was signed by no one other than Alice
 - consequently that the message has not been modified
 - i.e. the message is a true and correct copy of the original
- The difference between digital signatures and conventional ones is that digital signatures can be mathematically verified
- A digital signature scheme (or mechanism) consists of
 - a **signature generation algorithm**
 - a method for producing a digital signature
 - a **signature verification algorithm**
 - a method for verifying that a digital signature is authentic (i.e., was indeed created by the specified entity)



Digital Signature (cont.)

- Two general classes of digital signature schemes:
 - **Digital signature schemes with appendix**
 - require the original message as input to the verification algorithm
 - **Digital signature schemes with message recovery**
 - do not require the original message as input to the verification algorithm
 - in this case, the original message is recovered from the signature itself
- A digital signature scheme is said to be:
 - **deterministic**
 - signing operation is a one message-to-signature transformation
 - **randomized**
 - if the signing operation is a function also of a second parameter, leading to an indexing set of message-to-signature transformations

RSA Digital Signature with Appendix





RSA Signature

- PKCS#1 "RSA Cryptography Specifications Version 2.2" (RFC 8017) specifies two encoding methods for signatures with appendix:
 - **RSASSA-PKCS1-v1_5**
 - uses deterministic encoding
 - **RSASSA-PSS**
 - uses probabilistic encoding
 - includes a salt value
- These signature schemes combine signature/verification primitives with an encoding method for signatures
 - **a message encoding operation is applied to a message to produce an encoded message, which is then converted to an integer and processed by RSA signature primitive**
- Although no attacks are known against RSASSA-PKCS1-v1_5, RSASSA-PSS is preferred in new applications

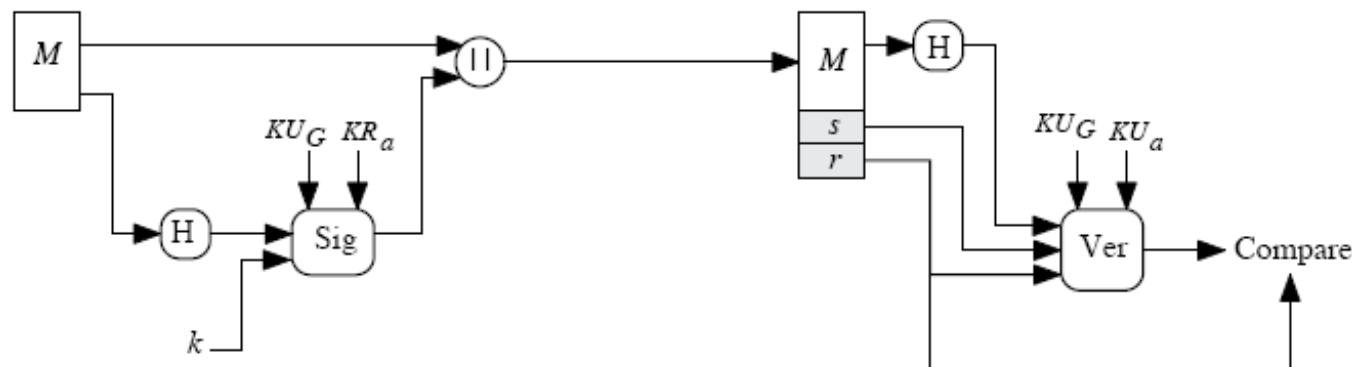
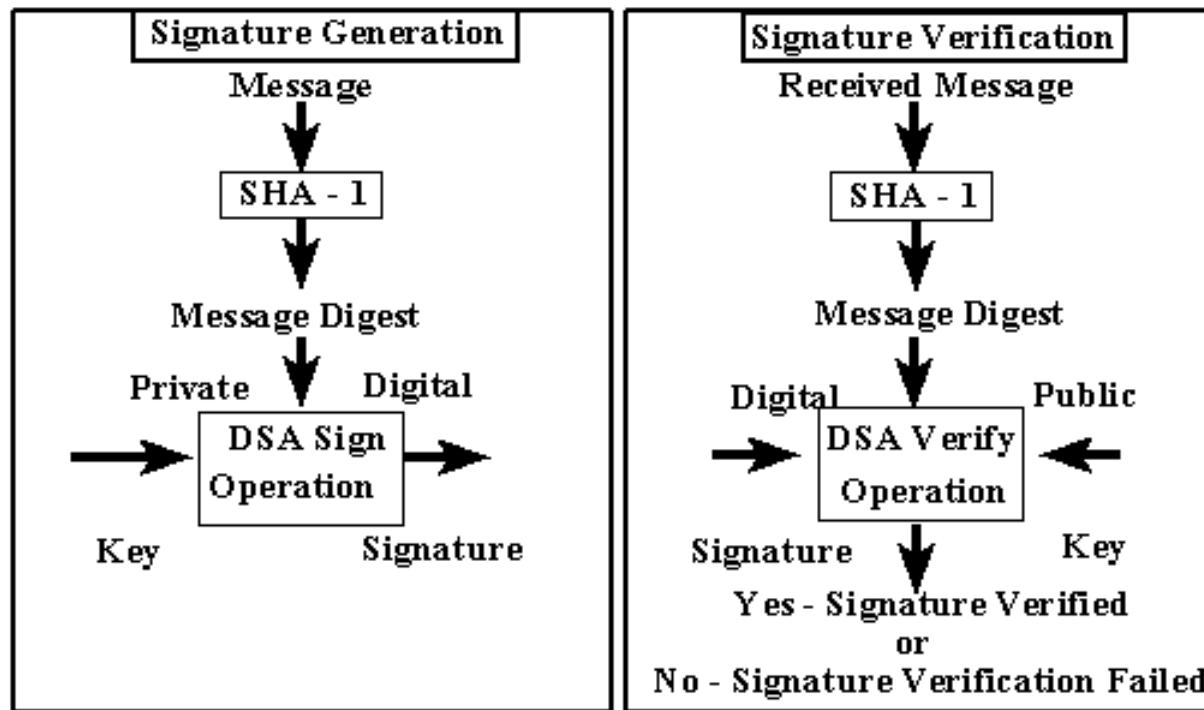


Digital Signature Standard (DSS)

- DSS (Digital Signature Standard)
- Proposed by NIST (U.S. National Institute of Standards and Technology) & NSA in 1991
 - FIPS 186
- Based on an algorithm known as DSA (Digital Signature Algorithm)
 - is a variant of the Elgamal (Taher Elgamal) scheme
 - uses hash algorithm, size N (e.g. SHA1, size 160)
 - uses N -bit exponents
 - creates a $2N$ bit signature but with 1024 (or more) bit security
- Security depends on difficulty of computing discrete logarithms



DSS Operations



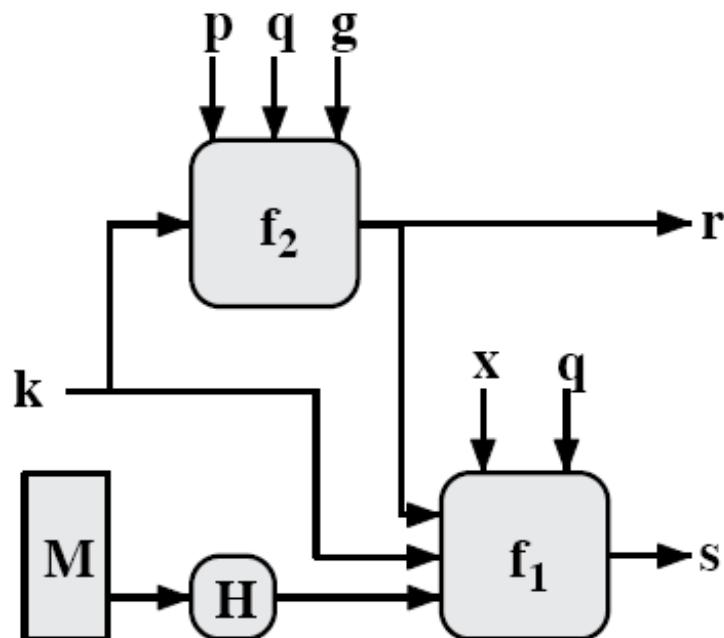


DSA Key Generation

- have shared global public key values (p, q, g)
 - **L and N are respectively the key length and hash length**
 - $L = 1024$ or more, and multiple of 64 (e.g. 1024, 2048, 3072,...)
 - $N = 160$ or more (e.g. 160, 256, ..)
 - **take a large (L -bit) prime p**
 - **choose q , a N -bit prime factor of $p-1$**
 - in practice, you can choose q , and then p such that $(p-1)$ is multiple of q
 - **choose g such that its multiplicative order modulo p is q**
 - in practice, $g = a^{(p-1)/q} \bmod p$
 - for some arbitrary a with $1 < a < p-1$, with $a^{(p-1)/q} \bmod p > 1$
- choose $x < q$
- compute $y = g^x \bmod p$
- public key = (p, q, g, y)
- private key = x

DSS Signing and Verifying schemes

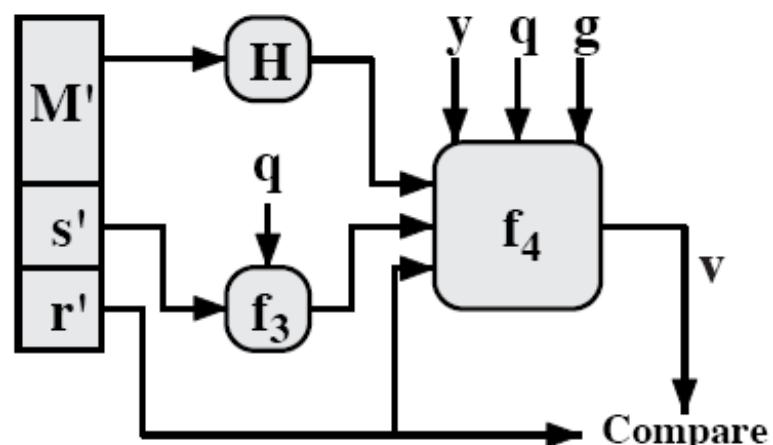
- Signing



$$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$$

$$s = f_1(H(m), k, x, r, q) = (k^{-1}(H(m) + xr)) \bmod q$$

- Verifying



$$w = f_3(s, q) = s^{-1} \bmod q$$

$$\begin{aligned} v &= f_4(p, q, g, y, H(m), w, r) = \\ &= ((g^{H(m)} w \bmod q)^{r w} \bmod q) \bmod p \bmod q \end{aligned}$$



DSA Signature Creation

- to sign a message m the sender generates:
 - a random signature key k , $k < q$
 - N.B.: k must be random, be destroyed after use, and never be reused
- computes the message digest (e.g. SHA-1) of the message m :
$$h = H(m)$$
- then computes signature pair:
$$r = (g^k \bmod p) \bmod q$$

$$s = k^{-1} (h + x \cdot r) \bmod q$$
- sends signature (r, s) with message m



DSA Signature Verification

- having received m & signature (r, s)
- to verify a signature, recipient computes:

$$w = s^{-1} \bmod q$$

$$v = (g^{hw} \bmod q \cdot y^{rw} \bmod q \bmod p) \bmod q$$

- if $v=r$ then signature is verified
- proof

$$v = (g^{hw} \bmod q \cdot y^{rw} \bmod q \bmod p) \bmod q =$$

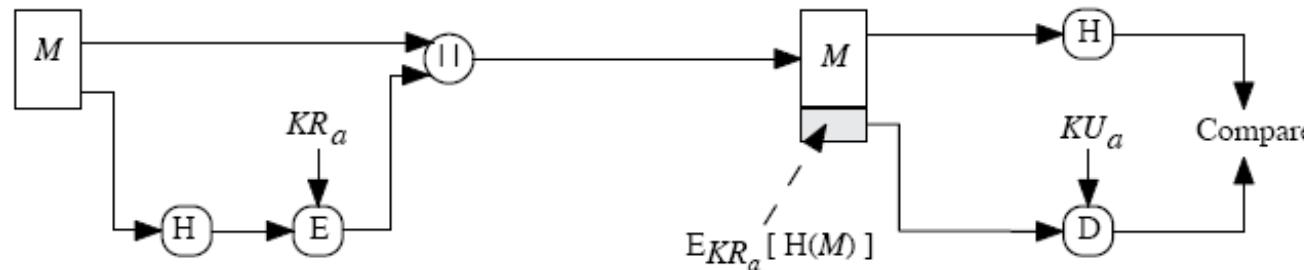
$$= (g^{w(h+xr)} \bmod q \bmod p) \bmod q =$$

$$= (g^k \bmod p) \bmod q =$$

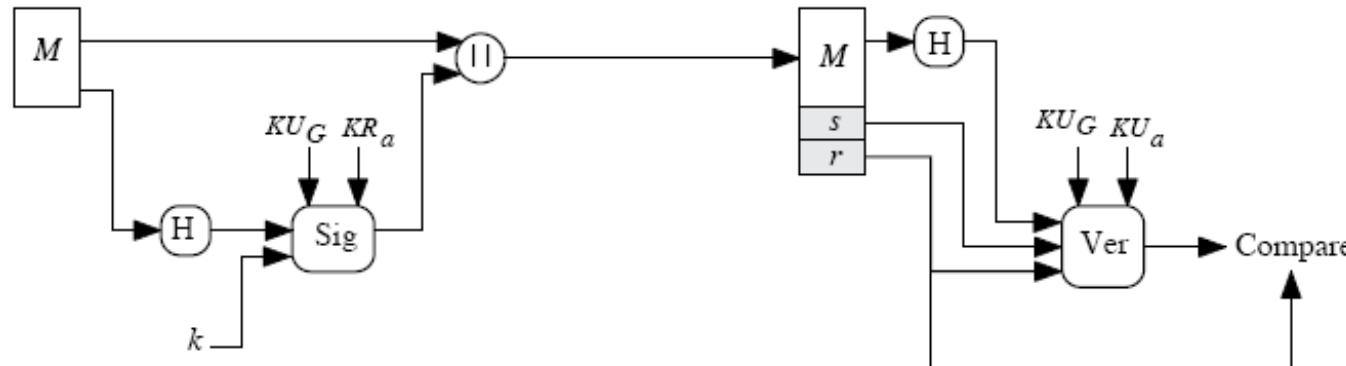
$$= r$$

RSA vs. DSS signatures

- RSA signature



- DSS signature





UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Identification (Entity Authentication)

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Entity Authentication - Identification

- Techniques designed to allow one party (the verifier or authenticator) to gain assurances that the identity of another (the prover, claimant, or supplicant) is as declared
 - **preventing impersonation**
- A major difference between entity authentication and message authentication is that
 - **message authentication itself provides no timeliness guarantees with respect to when a message was created**
 - **whereas entity authentication involves proof of a claimant's identity through actual communications with an associated verifier during execution of the protocol itself**
 - provide assurances only at the particular instant in time of successful protocol completion
 - If ongoing assurances are required, additional measures may be necessary



Basis of identification

- Entity authentication techniques may be divided into three main categories, depending on which of the following the security is based:
 - **1. something known**
 - e.g. standard passwords (sometimes used to derive a symmetric key), Personal Identification Numbers (PINs), secret or private keys
 - **2. something possessed**
 - this is typically a physical accessory, as a “passport”
 - e.g. cards like magnetic-striped cards, chip cards (also called smart cards or IC cards), and hand-held customized calculators (passwd generators) which provide time-variant passwords
 - **3. something inherent to a human individual**
 - this category includes methods which make use of human physical characteristics and involuntary actions (biometrics), such as handwritten signatures, fingerprints, voice, retinal patterns, and dynamic keyboarding characteristics
 - these techniques are not further discussed



Characteristics of identification protocols

- Direction of the identification (reciprocity):
 - **unilateral identification**
 - only one party proves its identity to the other party
 - **mutual identification**
 - both parties may corroborate their identities to the other
- Computational and/or communication efficiency:
 - **computational efficiency**
 - the number of operations required to execute a protocol
 - **communication efficiency**
 - this includes the number of passes (message exchanges) and the bandwidth required (total number of bits transmitted)

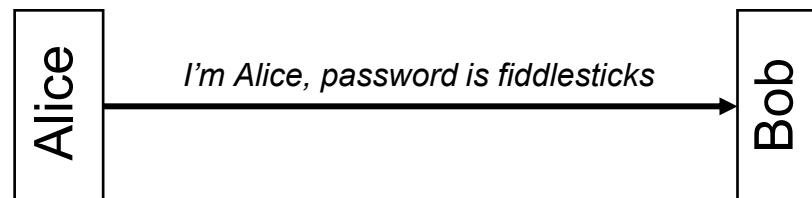


Characteristics of identification protocols (cont.)

- Real-time involvement of a third party (if any):
 - an on-line trusted third party to distribute symmetric keys to communicating entities for authentication purposes; or
 - an on-line trusted third party to verify the authentication information sent by the supplicant, or
 - an on-line (untrusted) directory service for distributing public-key certificates
- Storage of secrets
 - **which information must be stored to perform verification**
 - symmetric secret
 - in clear or encrypted/hashed value
 - in case of public keys, they are not secret, however:
 - how obtaining public key of the peer-entity
 - how storing public key of the peer-entity
 - **where secrets are maintained/stored**
 - RAM
 - local disks
 - hardware tokens

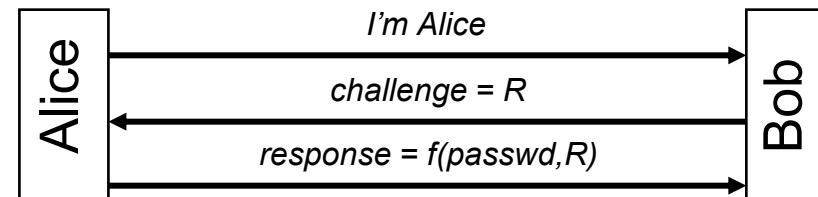
Basic authentication scheme

- Conventional (basic) authentication schemes consist in sending a fixed (time-invariant) symmetric secret
 - **shared secret between the user and system**
 - like a key or a password
 - **thus fall under the category of symmetric-key techniques**
- For example, to gain access to a system resource the user enters a (user id, password) pair
 - **weak authentication if used through an insecure channel**
 - **it is not associated to a given time (time-invariant)**
 - **the verifier can store the hash of the secret**

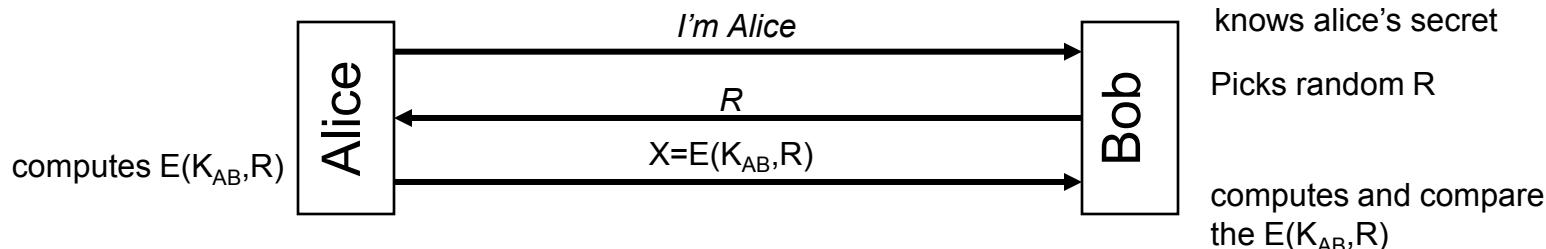


Challenge-response authentication

- The idea is that one entity (the claimant) “proves” its identity to another entity (the verifier) by demonstrating knowledge of a secret, without revealing the secret itself to the verifier
- This is done by providing a response to a *time-variant* challenge, where the response depends on both the challenge and the entity’s secret
 - ***time-variant* challenge is used to counteract replay and interleaving attacks**
 - a number, a text string, a bit string chosen randomly by one entity
 - a sequence number, incremented sequentially, a timestamp referring to a given time interval, optionally sent during the exchange
 - a validity interval of numbers may be considered
 - **the response is verified by using the same entity’s secret OR other information associated to the secret**
 - e.g. hash value, public key, etc.



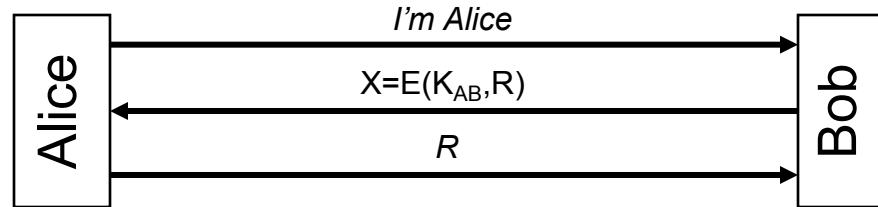
Authentication with symmetric key



- Note:
 - does not require reversible cryptography
 - function *E* can be replaced by one-way function of a secret and the challenge
 - e.g. $X=MAC(K_{AB},R)$, or $X=H(secret_{AB} \parallel R)$
 - in general: $X=f(K_{AB},R)$
- drawbacks:
 - an eavesdropper could mount an off-line password guessing attack
 - It requires that the authenticator maintain a copy of the password/key
 - some who read the Bob's passwd-database (the authenticator) can later impersonate Alice (the claimant)

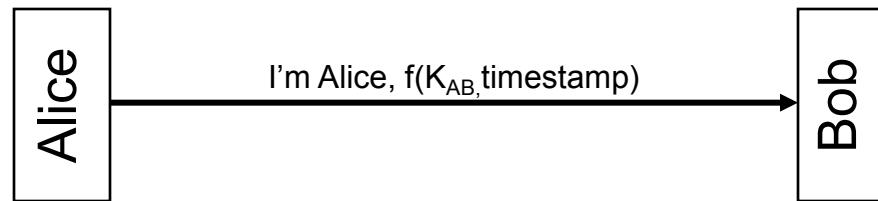


Authentication with symmetric key (variant 1)



- differences:
 - **requires reversible cryptography**
 - e.g. $R=D(K_{AB},X)$
 - **if R is a recognizable quantity with limited lifetime (e.g. a random number concatenated with a timestamp), Alice can authenticate Bob**
 - **if R is a recognizable quantity, Carol can mount an offline password-guessing attack without eavesdropping**
 - Carol obtains $K_{AB}\{R\}$ (second message) by just sending the first message to Bob claiming to be Alice
 - Carol doesn't need that Alice authenticates with Bob

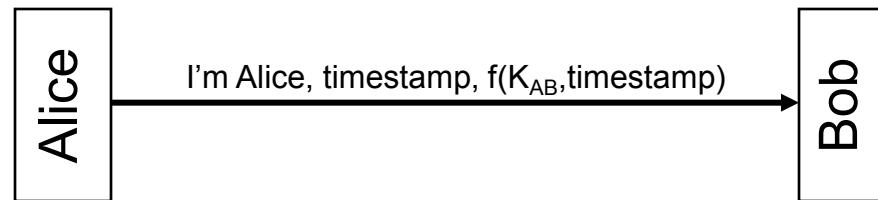
Authentication with symmetric key (variant 2)



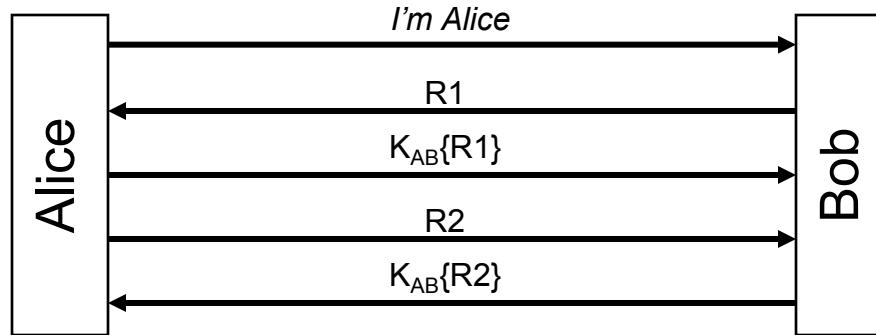
- differences:

- **required only one message**
 - this mechanism can be added very easily to a protocol originally designed for sending passwd as cleartext
 - more efficient
- **function f() does not require to be reversible**
- **several pitfalls due to the time validity
(time synchronization between Alice and Bob, authentication with multiple server with the same passwd, etc)**

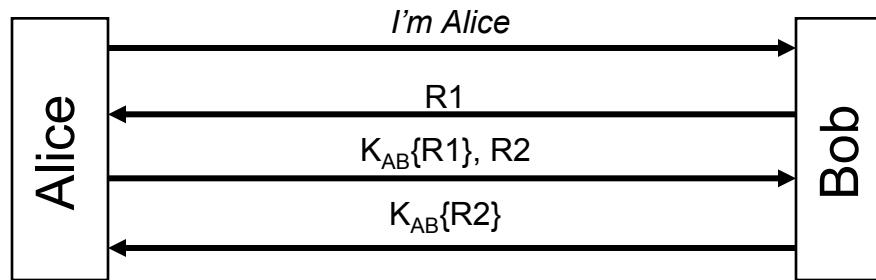
- variant:



Mutual authentication with symmetric key

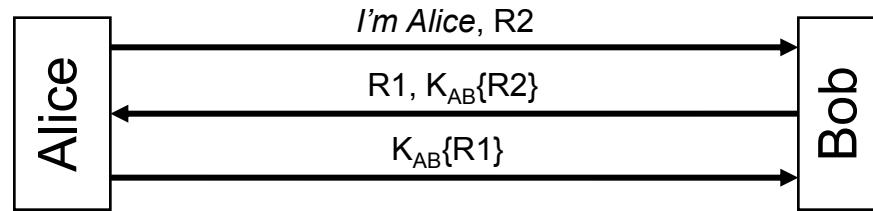


- or shorter:

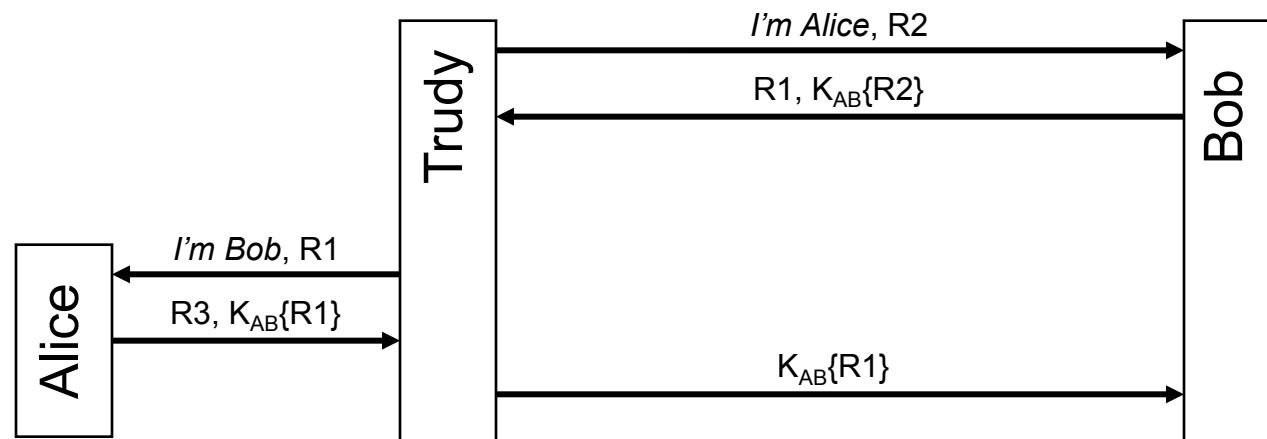


Mutual authentication with symmetric key

- or shorter:

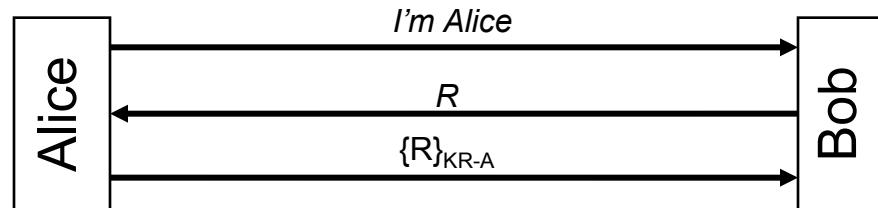


- however, possible reflection attack:

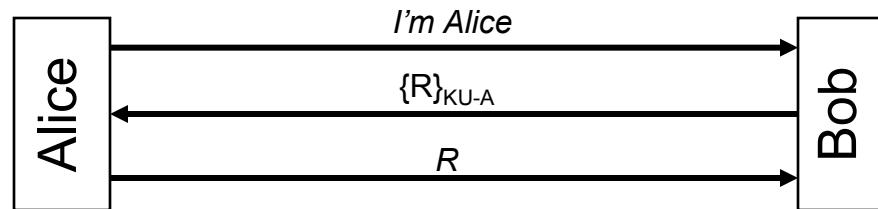


- Good general principles of authentication protocols:
 - the initiator should be the first to prove its identity
 - messages sent in opposite directions should differ

Authentication with public key



or



- property:
 - the database at Bob must not be protected from reading
 - must be protected only for unauthorized modification (integrity protection)
- drawbacks:
 - if you can trick Alice into signing or decrypting something, you can impersonate Alice (first and second scheme, respectively)
 - by asking Alice to authenticate, you can obtain a signature or decryption
- countermeasures:
 - not use the same key for two different purposes unless the design for all uses are coordinated (this is a general rule), and/or
 - impose enough structure to be signed (nonce, realm, timestamp, etc.)

Eavesdropping and server database reading

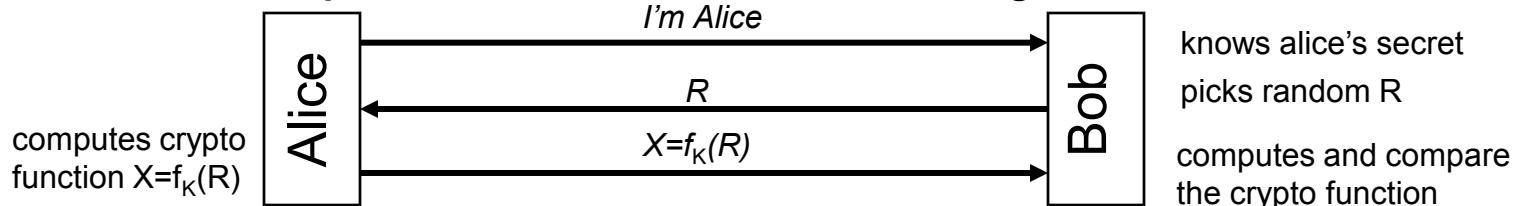
- Protection against server database reading:

- vulnerable to eavesdropping

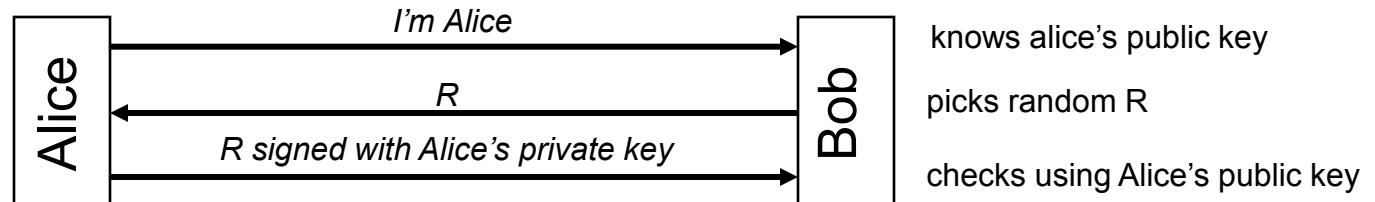


- Protection against eavesdropping:

- vulnerable to database reading, and to offline password guessing if the secret (key) is derived from a passwd, or offline brute force searching



- Protection against both using asymmetric cryptography:





One-time passwords

- An alternative to use fixed secrets/passwords is using one-time secrets/passwords
 - **each password is used only once**
 - **such schemes are safe from passive adversaries who eavesdrop and later attempt impersonation**
- Can be easily implemented in Smart/token Cards





One-time passwords (cont.)

- Some one-time passwords variations:

- **shared lists of one-time passwords**

- use a sequence or set of secret passwords, (each valid for a single authentication), distributed as a pre-shared list
 - if the list is not used sequentially, the system may check the entered password against all remaining unused passwords
 - a variation involves use of a challenge-response table
 - a drawback is maintenance of the shared list

- **sequentially updated one-time passwords**

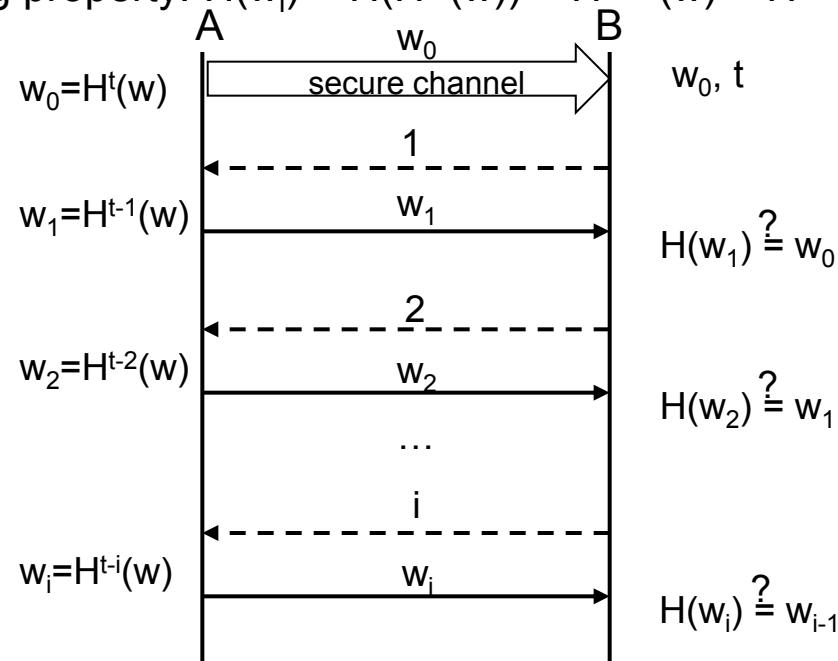
- during authentication using password i , the user creates and transmits to the system a new password (password $i+1$) encrypted under a key derived from password i
 - this method becomes difficult if communication failures occur

- **one-time password sequences based on a one-way function**

- more efficient than the previous one
 - may be viewed as a challenge-response protocol where challenge is implicitly defined by the current position within the pwd sequence

Lamport's scheme

- Simple One-Time Password (OTP) mechanism
 - does not require the server to maintain a shared secret nor a list of hashes
 - based on a secret w and a one-way function H , the sequence of t passwords $H(w), H(H(w)), \dots, H^t(w)$ is defined
 - these passwords are used in the reverse order
 - the authenticator is initialized with $w_0 = H^t(w)$
 - password for the i^{th} identification exchange ($1 \leq i \leq t$) is: $w_i = H^{t-i}(w)$
 - resulting property: $H(w_i) = H(H^{t-i}(w)) = H^{t-i+1}(w) = H^{t-(i-1)}(w) \equiv w_{i-1}$





Lamport's scheme (cont.)

- Lamport's scheme usage:
 - user A begins with a secret w and a constant t (e.g., $t = 100$ or 1000), defining the number of identifications to be allowed
 - A transfers (the initial shared secret) $w_0 = H^t(w)$, in a manner guaranteeing its authenticity, to the system B
 - B initializes its counter i_A for A to 1 ($i_A = 1$)
 - $A \rightarrow B : A, i, w_i = H^{t-i}(w)$
 - w_i is easily computed either from w or from an appropriate intermediate value saved during the computation of $H^t(w)$ initially
 - B checks that $i = i_A$, and that the received password w_i satisfies: $H(w_i) = w_{i-1}$



Zero-knowledge identification protocols

- Zero-knowledge (ZK) protocols allow a prover to demonstrate knowledge of a secret while revealing no information whatsoever
 - **beyond what the verifier was able to deduce prior to the protocol run**
- ZK protocols are often instances of interactive proof system, wherein a prover and verifier exchange multiple messages (challenges and responses), typically dependent on random numbers
- Example
 - **Fiat-Shamir identification protocol**



Basic Fiat-Shamir identification scheme

- It allows one party, Peggy (P), to prove to another party, Victor (V), that she possesses secret information without revealing the secret to V
 - **asymmetric cryptography identification scheme**
 - **it uses modular arithmetic**
- Setup
 - a RSA-like modulus $n = pq$, is selected and published by the claimant P or by a trusted center T selects, while primes p and q are kept secret
 - each P selects a secret $s < n$ coprime to n , computes $v = s^2 \bmod n$, and publishes v (or v is sent to V)
- Procedure
 - each of t rounds has three messages as follows
 - $P \rightarrow V : x = r^2 \bmod n$ (witness)
 - $P \leftarrow V : c \in \{0,1\}$ (challenge)
 - $P \rightarrow V : y = r \cdot s^c \bmod n$ (response)
- Verification (each round):
 - V verifies that $y^2 = x v^c \bmod n$



Fiat-Shamir identification scheme (cont.)

- Explanation:
 - the challenge (or exam) c requires that P is capable of answering two questions, one of which demonstrates her knowledge of the secret s , and the other an easy question (for honest provers) to prevent cheating
 - an adversary impersonating P might try to cheat by selecting any r and setting $x = r^2/v$, then answering the challenge $c = 1$ with a “correct” answer $y = r$; but would be unable to answer the exam $c = 0$ which requires knowing a square root of $x \bmod n$
 - a prover P knowing s can answer both questions, but otherwise can at best answer one of the two questions, and so has probability only $1/2$ of escaping detection
 - by iterating the protocol t times (e.g., $t = 20$ or $t = 40$), the probability of cheating decreases to an arbitrary acceptable small value $(1/2)^t$
 - V accepts P 's identity only if all t questions (over t rounds) are successfully answered
- Security
 - the security relies on the difficulty of extracting square roots modulo large composite integers n of unknown factorization, which is equivalent to that of factoring n



Authentication attacks

- Possible authentication attacks are:
 - **Impersonation attack**
 - pretend to be client or server
 - **Replay attack**
 - a valid message is copied and later resent
 - **Reflection attack**
 - re-send the authentication messages elsewhere
 - **Modify attack**
 - modify messages between client and server
 - **Compromising of key material**
 - steal client/server authentication database



Replay and reflection countermeasures

- Countermeasures against replay and reflection attacks include
 - **use of sequence numbers**
 - difficult to implement in practice
 - **use of timestamps**
 - needs synchronized clocks
 - **use of challenge/response**
 - using unique nonce, salt, realm values

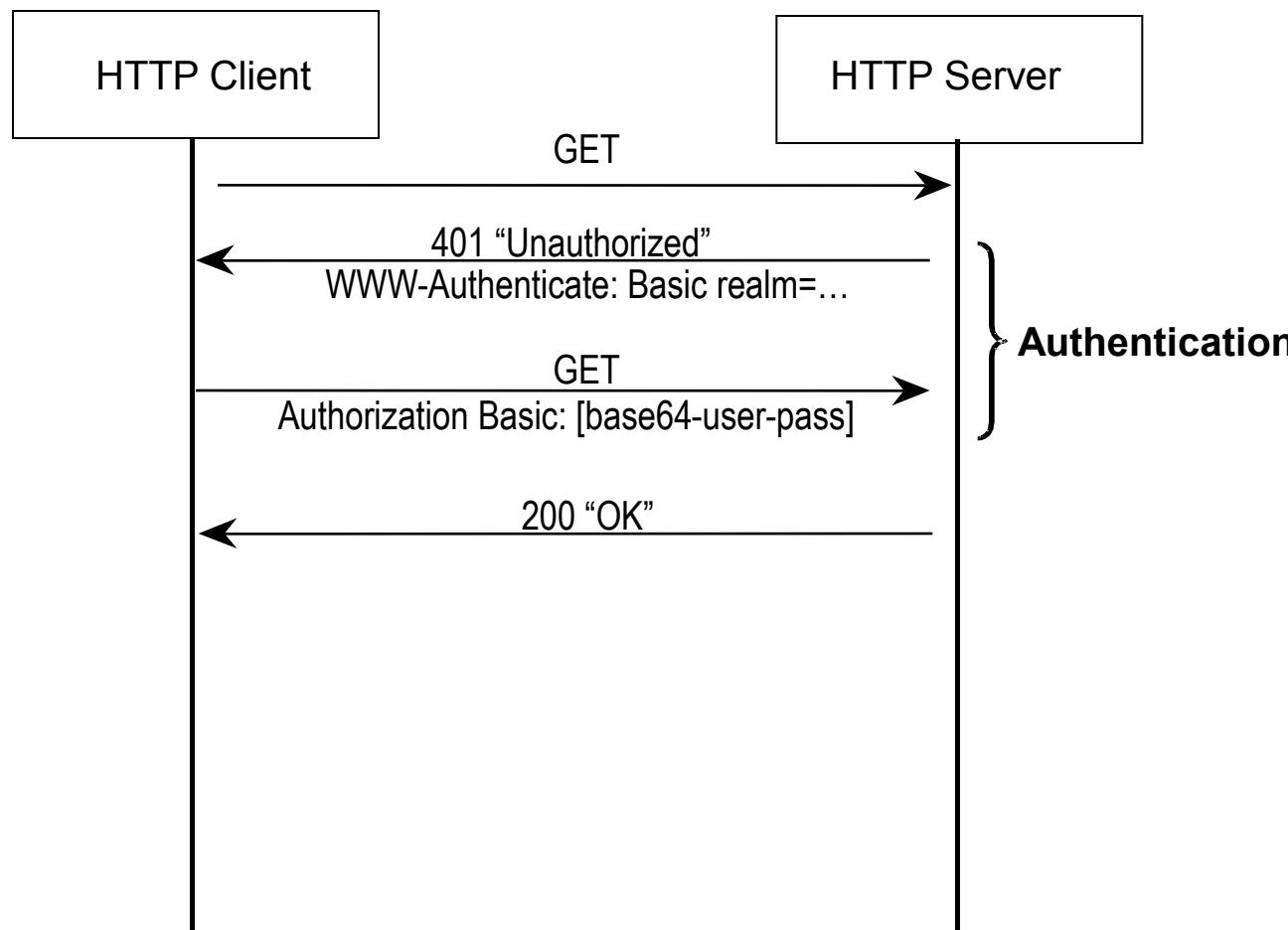
Examples of authentication protocols



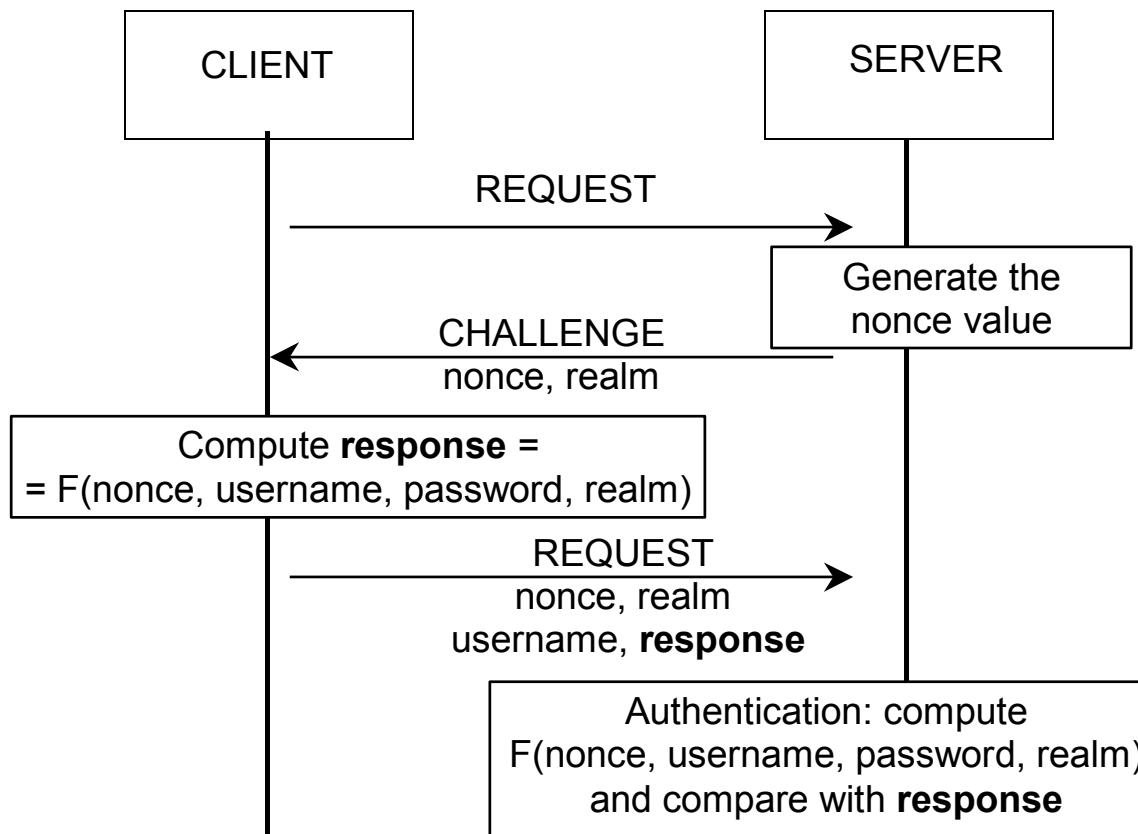
HTTP Basic and Digest Authentication

- Any time that a HTTP server receives a request, it MAY challenge the initiator of the request to provide assurance of its identity
- Two different types of client authentication schemes:
 - “**basic**” authentication
 - the client must authenticate itself with a user-ID and a password for a given realm
 - this scheme is not considered to be a secure method of user authentication as the user name and password are passed in an unencrypted form (unless secure transport is used)
 - “**digest**” authentication
 - based on a simple stateless challenge-response paradigm
 - the server challenges the client using a nonce value
 - a valid response contains a checksum (by default, through MD5) of the username, the password, the given nonce value, the HTTP method, and the requested URI
 - message authentication and replay protection
 - used also by other HTTP-based protocol (e.g. SIP for VoIP)

HTTP Basic authentication - Example

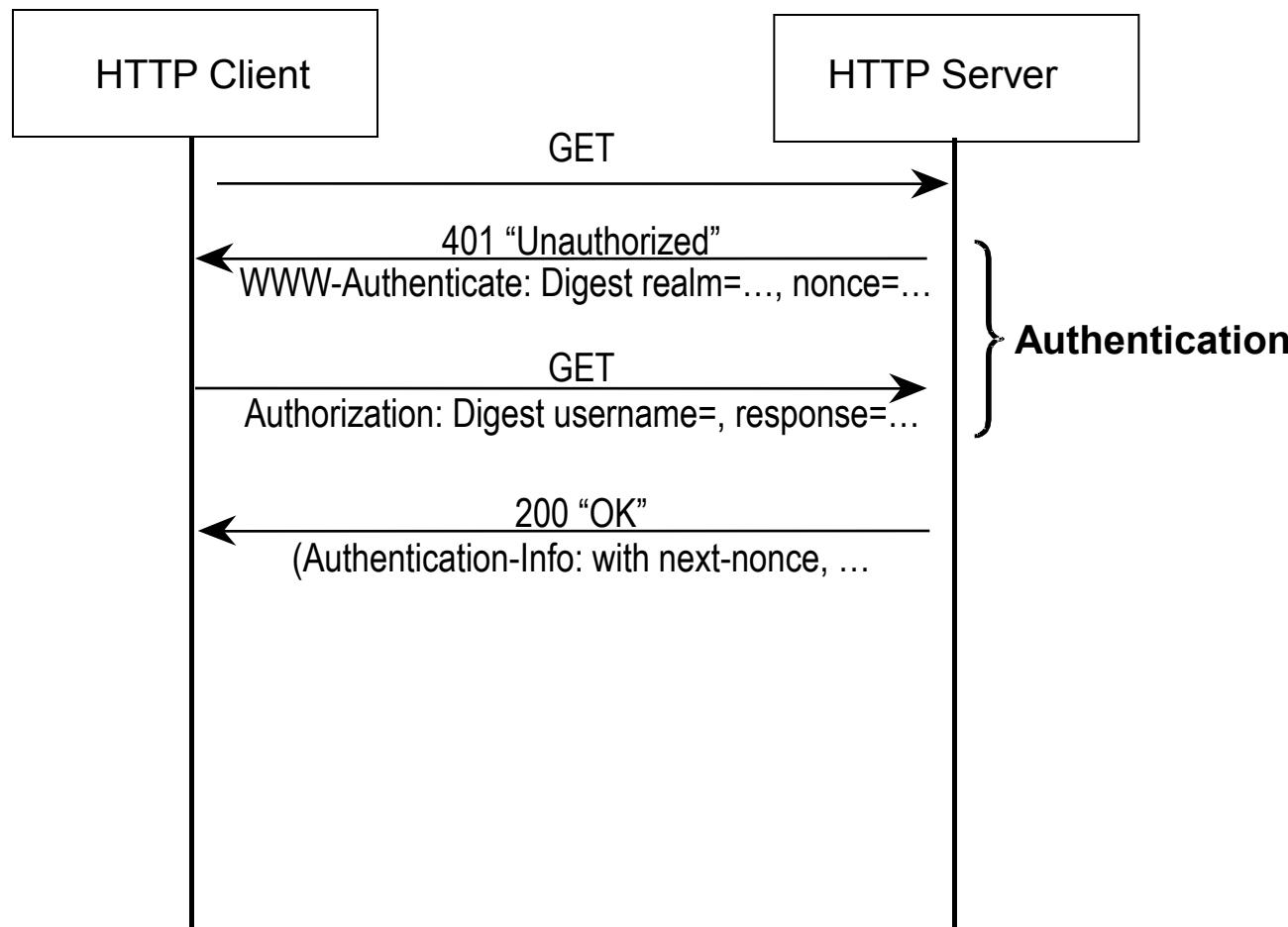


HTTP Digest authentication (cont.)





HTTP Digest authentication - Example



HTTP Digest authentication - Example (cont.)

- 401 Unauthorized response message:

```
WWW-Authenticate: Digest realm="biloxi.com", qop="auth,auth-int",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    algorithm=MD5
```

- Next request message:

```
Authorization: Digest username="bob", realm="biloxi.com",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    uri="/dir/index.html", qop=auth,
    response="6629fae49393a05397450978507c4ef1"
```

➤ **response = F(nonce, username, passwd, realm, metod, http uri)**



Digest calculation details

- If the "qop" value is "auth" or "auth-int", the F() digest value is computed as follows:
 - **If the "qop" directive's value is "auth" or is unspecified, then A2 is:**
A2 = Method : digest-uri-value
 - **If the "qop" value is "auth-int", then A2 is:**
A2 = Method : digest-uri-value : H(entity-body)
 - **If the "algorithm" directive's value is "MD5" or is unspecified, then A1 is:**
A1 = username-value : realm-value : passwd
 - **then, F() is:**

H(H(A1) : nonce-value : nc-value : cnonce-value : qop-value : H(A2))



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Cryptography: Symmetric Key Establishment

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Key management

- In a secured communication, users must setup the details of the cryptography
 - In some cases this may require sharing a symmetric key
 - In others it may require obtaining the other party's public key
- In both cases, an issue is how to securely distribute these keys
 - **in case of symmetric key**
 - must be exchanged over a secure communication channel
 - both confidentiality and data authentication must be guaranteed
 - **in case of public key**
 - keys can be openly exchanged (authenticated) over an insecure communications channel
 - only message authentication must be guaranteed
 - » the exchange is less troublesome
 - » for example, by using digital certificates
- Often secure systems fail due to a break in the key distribution



Long-term vs Short-term Keys

- Cryptographic system and protocols usually deal with two types of keys with two different lifetimes:
 - **short-term keys**
 - also called as session keys
 - used to secure a communication
 - data confidentiality
 - data authentication and integrity
 - used only for a limited interval of time
 - setup through a proper KE protocol
 - **long-term keys**
 - also referred to as long-term secrets
 - can be used for
 - peer authentication
 - securing key exchanges



(Secret) Key establishment

- Secret key establishment is a process or protocol whereby a shared secret becomes available to two or more parties, for subsequent cryptographic use
- Two approaches are possible:
 - **Key pre-distribution**
 - key establishment schemes whereby the resulting established keys are completely determined “a priori” by initial keying material
 - used mainly only for long-term secret keys
 - **Dynamic key establishment (or key exchange)**
 - those schemes whereby the key established by a pair (or group) of users varies on subsequent executions
 - subdivided into (see later):
 - key transport
 - key agreement



Key establishment (cont.)

- Dynamic key establishment may be broadly subdivided into:
 - **key transport**
 - A key transport protocol or mechanism is a key establishment technique where one party creates or otherwise obtains a secret value, and securely transfers it to the other(s)
 - **key agreement**
 - A key agreement protocol or mechanism is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of these, (ideally) such that no party can predetermine the resulting value
- Additional variations exist, including various forms of key derivation and key update



Use of trusted servers

- Some key establishment protocols involve a centralized or trusted party, for either or both:
 - **initial system setup**
 - **on-line actions (involving real-time participation)**
- This party is referred to by a variety of names depending on the role played
 - **generically called as trusted third party or trusted server**
- E.g.
 - **authentication server**
 - **key distribution center (KDC)**
 - **certification authority (CA)**



Properties of key establishment

- Properties that a KE protocol may satisfy:
 - **Entity authentication**
 - a party in a key establishment protocol is able to determine the true identity of the other(s) which could possibly gain access to the resulting key
 - **Implicit Key authentication**
 - is the property whereby one party is assured that no other party aside from a specifically identified second party (and possibly additional identified trusted parties) may gain access to a particular secret key
 - the other protocol participant is the only other party that could possibly be in possession of the correct established key
 - **Key confirmation**
 - is the property whereby one party is assured that a second (possibly unidentified) party actually has possession of a particular secret key
 - **Explicit key authentication**
 - both implicit key authentication and key confirmation hold
 - an identified party is known to actually possess a specified key
- A KE protocol is usually called "Authenticated Key Exchange" (AKE) if both entity authentication and explicit key authentication are provided
- In addition:
 - **Key freshness assurance**
 - assurance that the exchanged key is new



Forward secrecy and known-key attacks

- It is important to consider the potential impact of compromise of various types of keying material
 - even if such compromise is not normally expected
- In particular, the effect of the following is often considered:
 - compromise of past session (established) keys
 - compromise of long-term secret (symmetric or asymmetric) keys
- A protocol is said to be "resistant to a known-key attack" if compromise of past session keys doesn't allow an adversary to compromise future session key exchanges
- A protocol is said to have "(perfect) forward secrecy" if compromise of long-term keys does not compromise past session keys
 - example: Diffie-Hellman key agreement, in some cases, may provide forward secrecy

Key transport and derivation using symmetric cryptography



Point-to-point key transport and key derivation

- Key transport or derivation based on a long-term symmetric key K_{ab} shared “a priori” by two parties A and B
 - the long-term key must be initially distributed over a secure channel or resulting from a key pre-distribution mechanism
 - the long-term key is used to establish new session keys K_s
 - KE protocol (transport/derivation): $K_{AB} \rightarrow K_s$
- Possible techniques:
 - key transport based on symmetric encryption
 - key derivation based on non-reversible functions
- Variants:
 - key transport with one pass
 - key transport with challenge-response



Key transport

- Key transport with one pass:

$$A \rightarrow B : E_{Kab}(K_S)$$

- both A and B obtain implicit key authentication
- susceptible to known-key attacks through replay attack

- Additional optional fields might be transferred in the encrypted portion:

$$A \rightarrow B : E_{Kab}(K_S, t_A^*, B^*)$$

- field containing redundancy provides explicit key authentication to B and facilitates message modification detection
- timestamp (or sequence number) provides a sort of freshness guarantee to B
 - avoids replay attacks
- a destination identifier prevents message replay back on A
 - if a timestamp is present it provides also entity authentication to B



Key transport (cont.)

- Key transport with challenge-response:
 - If anti-replay, explicit key authentication, and entity authentication are desired for B but reliance on timestamps added by A is not, a random value or sequence number n_B may be used
 - the cost is an additional message
- If it is required that the session key K_S be a function of inputs from both parties and mutual authentication:

$$A \leftarrow B : n_B$$
$$A \rightarrow B : E_{Kab}(K_S, n_B, B^*)$$
$$A \leftarrow B : n_B$$
$$A \rightarrow B : E_{Kab}(k_1, n_A, n_B, B^*)$$
$$A \leftarrow B : E_{Kab}(k_2, n_B, n_A, A^*)$$

➢ $K_S = f(k_1, k_2)$



Key transport (cont.)

- Vulnerabilities:
 - Any previous exchange does not offer forward secrecy
 - they fail if the long-term key K_{ab} is compromised
 - for this reason they may be inappropriate for many applications
- Note:
 - Authentication protocols which employ encryption, including the above key exchange schemes, may require that the encryption function has a built-in data integrity mechanism to detect message modification
 - i.e. Authenticated Encryption (AE)



Key derivation

- Key exchange may be achieved also by dynamic key derivation where the derived session key is based on per-session random input provided by one party

$A \rightarrow B : r_A$

- **single message**
- **the session key is computed as $K_s = f_{Kab}(r_A)$**
 - where f is a cryptographic function, like $E_K()$ or $MAC_K()$
- **provides to both A and B implicit key authentication**
- **susceptible to known-key attacks through replay attack**
- The random number r_A here may be replaced by other time-variant parameters
 - **e.g. a timestamp t_A , provides an implicit key freshness property**
 - avoids replay attack to B
 - avoids A can force a given key X



Key derivation (cont.)

- In general is preferred a keyed one-way function (like $\text{MAC}_K()$) for $f_K()$, in place of an encryption function $E_K()$
 - A cannot control the value of K_s
- The simple key derivation scheme can be further extended with two or three passes in order to provide:
 - contribution of B to the creation of K_s
 - mutual entity authentication



Example of authenticated key derivation protocol

- Authenticated Key Exchange Protocol AKEP2
 - provides mutual entity authentication, key freshness guarantee, and implicit key authentication
 - A and B exchange 3 messages
- *Setup:*
 - A and B share long-term symmetric keys K_{ab}

$A \rightarrow B : r_A$

$A \leftarrow B : X_B = \{B, A, r_A, r_B\}, \text{MAC}_{Kab}(X_B)$

$A \rightarrow B : X_A = \{A, r_B\}, \text{MAC}_{Kab}(X_A)$

- $K_s = f_{K'}(r_B)$
 - where $f_{K'}$ () is a keyed one-way function or encryption function
 - K' is the key used to derive K_s , usually obtained from K_{ab}

Key transport using asymmetric cryptography



Key transport based on public-key encryption

- One party may choose a symmetric key and transfer it to a second, using that party's encryption public key

$$A \rightarrow B : \{ K_s \} KU_B$$

- **this provides (implicit) key authentication to the originator (A)**
 - only the intended recipient has the private key allowing decryption
 - the originator (A) obtains neither entity authentication nor key confirmation
- **the second party (B) has no assurances regarding the source of the key and the timeliness**
 - even if A sends to B: $\{K_s, A, T_A\} KU_B$, since KU_B is public
- **such additional assurances may be obtained through use of further techniques including:**
 - additional messages, with public-key encryption
 - using challenge-response like key transport based on symmetric key
 - digital signature



Key transport using public-key encryption and signature

- Encryption and signature primitives may be used to provide respectively:
 - **privacy of keying material**
 - **source authentication**
- Some possible approaches:
 - **sign the key and separately public-key encrypt the (unsigned) key**
 - **sign the key, then public-key encrypt the signed key**
 - **public-key encrypt the key, then sign the encrypted key**



Key transport using public-key encryption and signature in one-pass

- Encrypting and signing separately:

➤ An option is to sign the key and encrypt the key:

$$A \rightarrow B: \{A, K_s, t_A^*\}KU_B, \text{Sign}_A(B, K_s, t_A^*)$$

- Encrypting signed keys:

➤ A variation is to encrypt signed blocks:

$$A \rightarrow B: \{A, K_s, t_A^*, \text{Sign}_A(B, K_s, t_A^*)\}KU_B$$

- Signing encrypted keys:

➤ In contrast to encrypting signed keys, one may sign encrypted keys

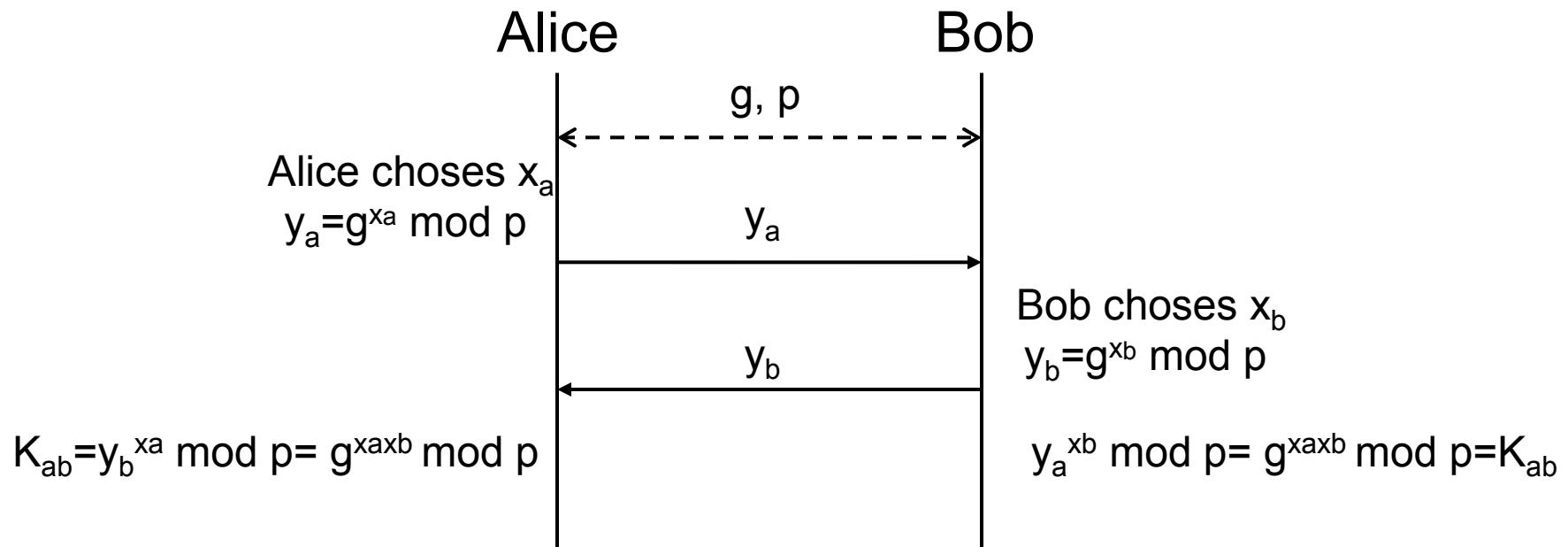
$$A \rightarrow B: t_A^*, Q, \text{Sign}_A(B, t_A^*, Q)$$

where $Q = \{A, K_s\}KU_B$

Key agreement using asymmetric cryptography

Diffie-Hellman key exchange

- The first publicly known public-key agreement protocol was the Diffie-Hellman exponential key exchange





DH modes

- Different ways of using DH:

- **Fixed Diffie-Hellman**

- each party already has the public part (DH public key) of the other entity

- **Anonymous Diffie-Hellman**

- DH exchange
 - susceptible to Man-in-the-Middle attacks

- **Ephemeral Diffie-Hellman (EDH)**

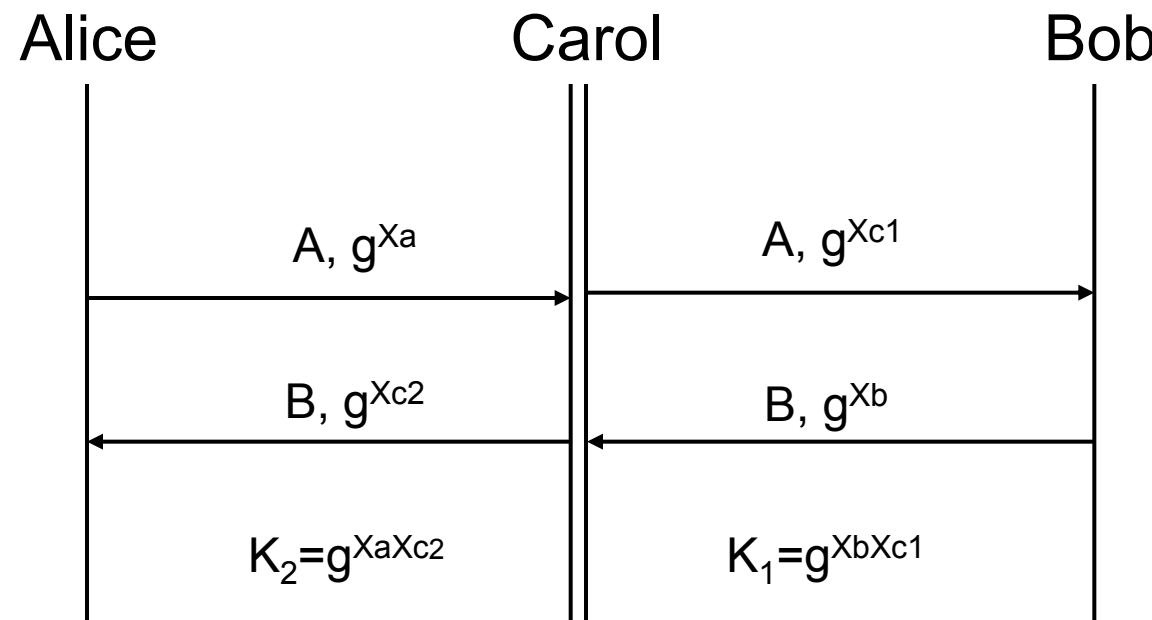
- authenticated DH exchange

- Elliptic Curve variant:

- **Elliptic Curve DH (ECDH) uses elliptic curves instead of the multiplicative group of integers modulo p**

Diffie-Hellman Vulnerability (MITM attack)

- Anonymous Diffie-Hellman key exchange, does not provide authentication of the parties, and is thus vulnerable to Man-in-the-middle attacks
 - a third party C may run two separate exchanges with A and B, convincing the two peers that the exchange ended successfully between them



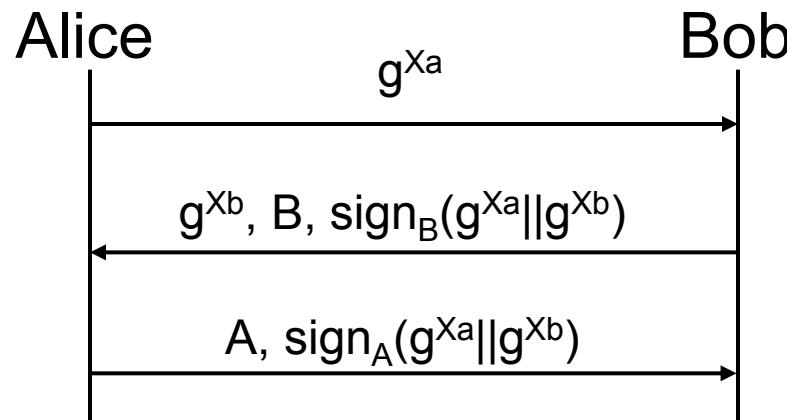


Authenticated DH Exchange

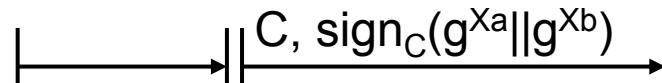
- A wide variety of schemes and protocols have been developed to provide authenticated DH key agreement to prevent man-in-the-middle and related attacks
- These methods generally use:
 - **Public/private key pairs**
 - **Shared secrets**

Authenticated DH Exchange (cont.)

- The simplest way to authenticate the DH exchange could be to sign the DH exponentials



- **It does not guarantee implicit key authentication nor confirmation**
 - the authenticated entity may be different from the peer that exchanged the secret key
 - a third party C can convince B that the exchange was run with C, by simply replacing the third message with:



- this attack does not result in a breach of secrecy of the key



Authenticated DH Exchange (cont.)

- Solution: uses signature and Enc

- variant of Station-to-Station (STS) protocol

A → B: g^{x_a}

A ← B: $g^{x_b}, E_{K_s}(B \parallel \text{Sign}_B(g^{x_a} \parallel g^{x_b}))$

A → B: $E_{K_s}(A \parallel \text{Sign}_A(g^{x_a} \parallel g^{x_b}))$

- Solution: uses signature and MAC

- SIGMA Protocol:

- signatures authenticate the DH exponentials
 - MACs bind the key to identities
 - MAC is performed with a key K_m derived by $K_s = g^{x_a x_b}$

A → B: g^{x_a}

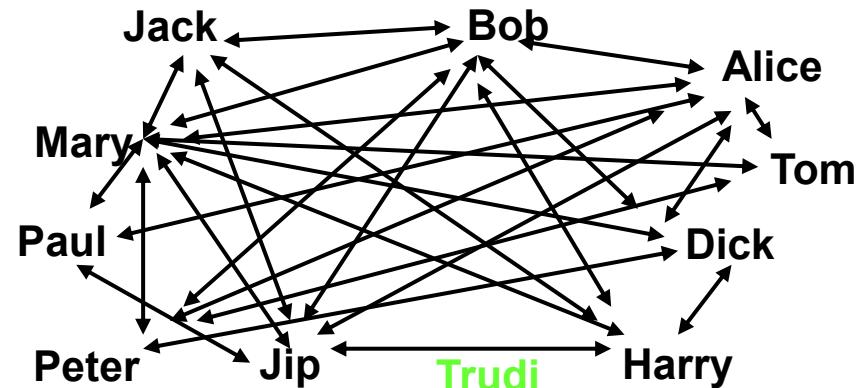
A ← B: $B, g^{x_b}, \text{Sign}_B(g^{x_a} \parallel g^{x_b}), \text{MAC}_{K_m}(B)$

A → B: $A, \text{Sign}_A(g^{x_a} \parallel g^{x_b}), \text{MAC}_{K_m}(A)$

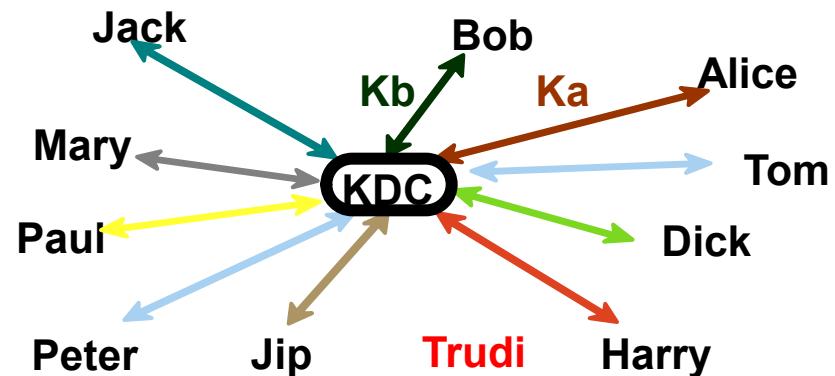
Server-based Key Distribution

Direct trust vs. Trusted intermediaries

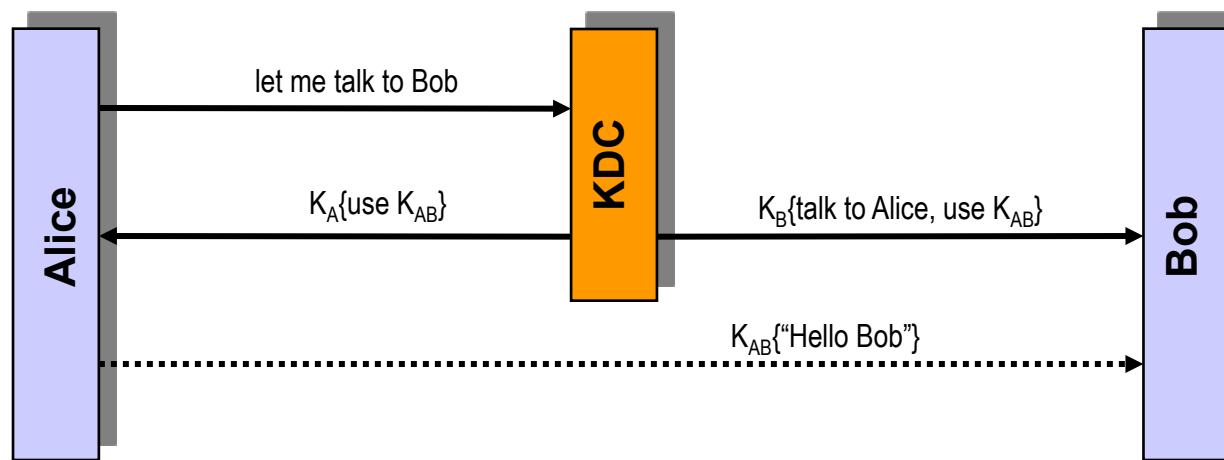
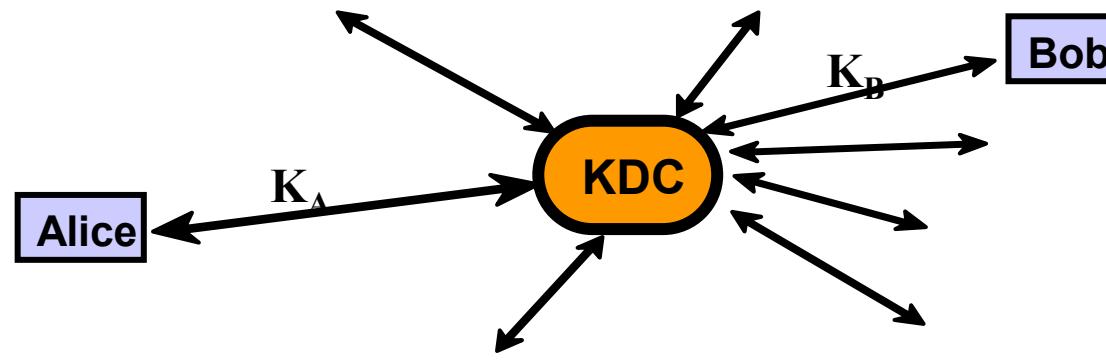
- With N nodes, each node must authenticate each other.. $N-1$ keys maintained by each node



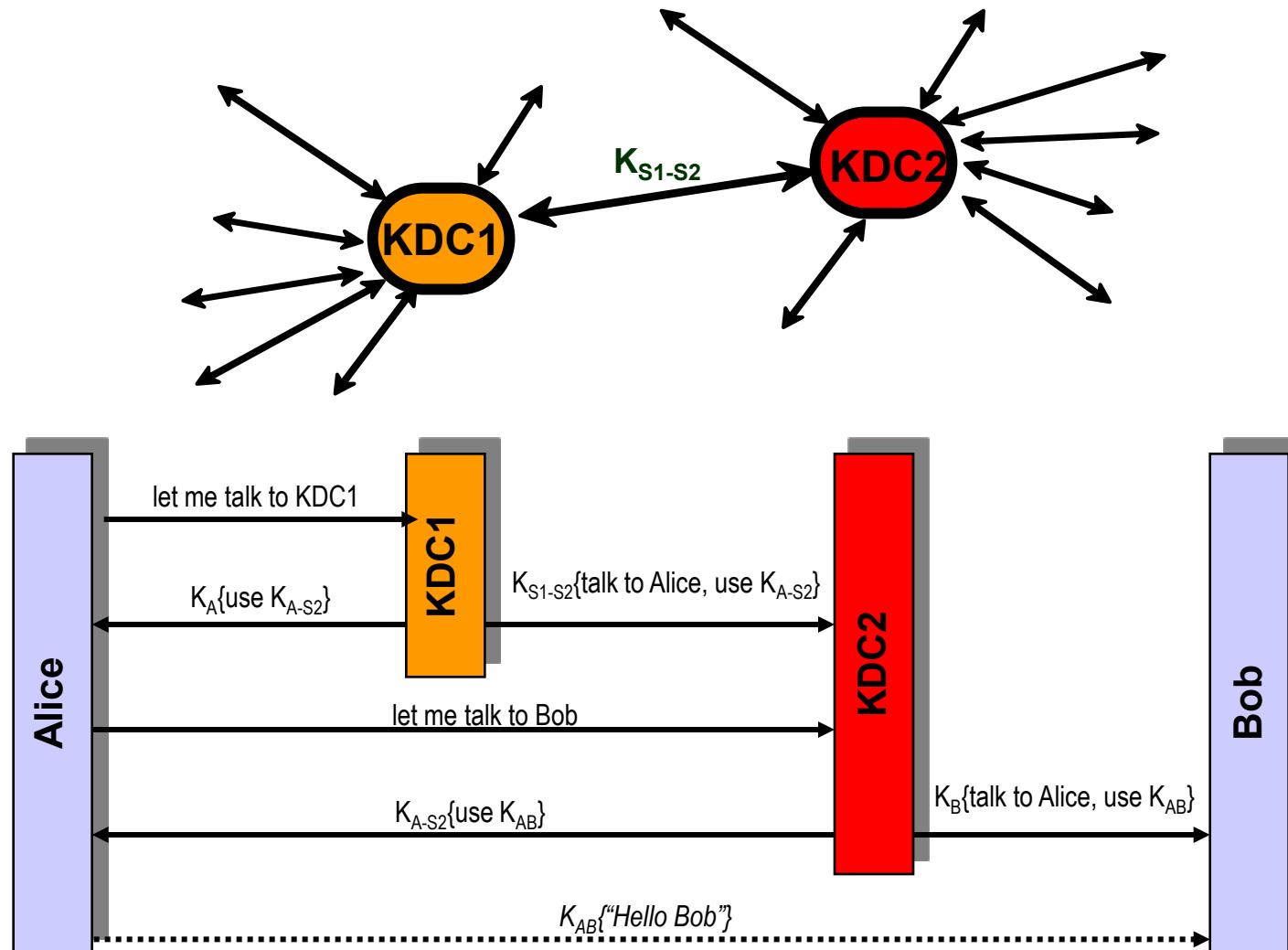
- Possible solution: Trusted intermediate party
 - Key Distribution Center (KDC) or Key Translation Center (KTC)
 - similar to CAs for public-key cryptography



Key distribution with a trusted intermediary

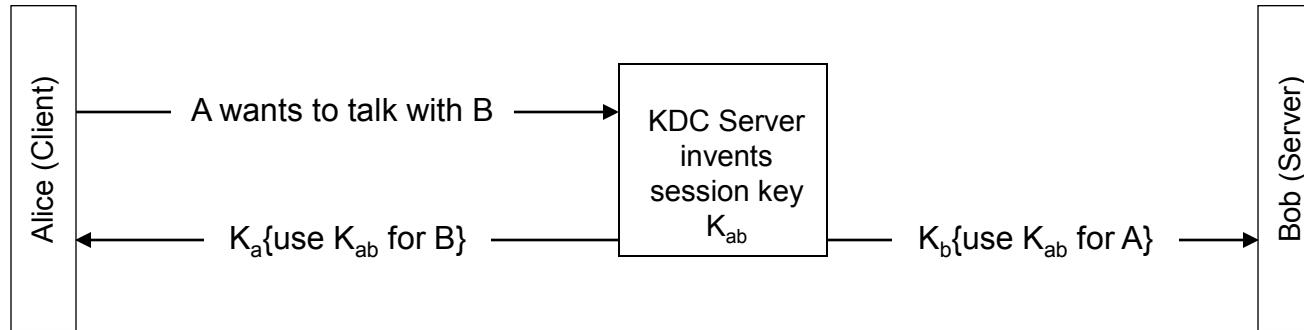


Key distribution with multiple trusted intermediaries

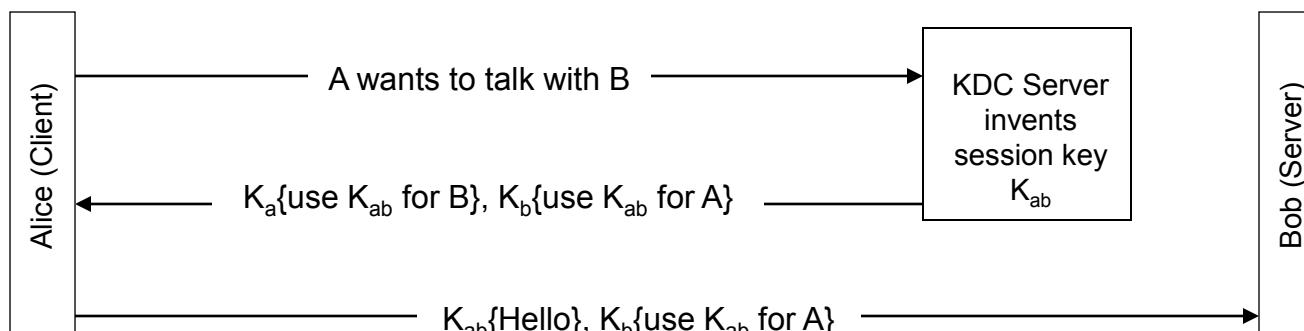


Key Distribution in practice

- Key Distribution in theory (Client-Server)



- Key Distribution in practice (Client-Server)
 - e.g. used by Kerberos protocol





UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Public key distribution and Digital Certificates

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Public Key Distribution

- Public key cryptography solves a major problem with symmetric algorithms
- 1) But how do you get my public key?
- 2) And how do you know it is my public key?
- In order to distribute public keys, the same scheme used for secret key distribution could be used, however
 - **the public key, used for encrypting or for verifying a signature, is not required to be secret**
 - publicly known key can be used, however
 - the key should be authenticated
- Digital certificates help to solve these two issues



Public Key Distribution (cont.)

- The simplest way is to use digital signature

$$C \rightarrow B : K_A^+, \text{sign}_{K_C}(K_A^+)$$

➤ **B must trust C and must know the C's public key**

- In general, the signing entity may be different from the entity that B receives the key from

$$D \rightarrow B : K_A^+, \text{sign}_{K_C}(K_A^+)$$

➤ **entity B receives from an entity D the public key of A signed by a third party C**

- In order to securely associate the K_A^+ to A, and the signature to C, the identities of A and C should also be included

$$D \rightarrow B : x = \{K_A^+ || ID_A || ID_C\}, \text{sign}_{K_C}(X)$$

➤ **the resulting data is called *digital certificate***

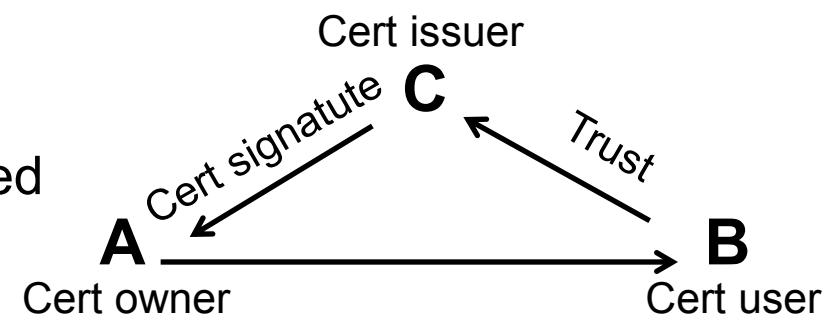
- In some practical cases it is directly A to send her cert to B

Digital Certificates

- Certificates are digital documents certifying the association of a public key with its owner
 - the certification is provided through a digital signature
 - $\text{cert}_C(A) = \{ X=\{K_A^+, ID_A, ID_C\}, \text{sign}_{K_C}(X) \}$
- A digital certificate may include:
 - a public key (K_A^+)
 - the name of the owner of the public key (ID_A)
 - the name of the issuer of the certificate (ID_C)
 - other certificate-related information:
 - a serial number
 - issuing and expiring date (validity)
 - other attributes
 - the digital signature of the issuer
 - $\text{sign}_{K_C}(\text{cert-data})$
- The signature is performed by a trusted third party C
 - e.g. a Certification Authority



Cert distribution
(e.g. through a network
or an untrusted third party D)





Digital Certificates (cont.)

- Certificates are normally used to exchange public keys in secure (authenticated) way, e.g.:
 - **for document signature**
 $A \rightarrow B : m, \text{sign}_{K_A}(m), \text{cert}_C(A)$
 - **for key distribution**
 $B \rightarrow A : \text{cert}_C(B)$
 $A \rightarrow B : \{ K_S \} K_B^+$
 - **for entity authentication**
- Certificates are usually deployed in
 - **Web transactions**
 - HTTPS uses TLS/SSL
 - **Virtual Private Networks**
 - IPSec uses IKE
 - **Secure messaging**
 - S/MIME and PGP
 - **Anywhere strong authentication and/or encryption is required**



Digital Certificates (cont.)

- The owner of the public key (and of the certificate) could be:
 - **a person**
 - **an alias of a person**
 - **an organization**
 - **a role within an organization**
 - **a hardware system**
 - **a software system**
- Some certificates are specific (and valid only) for a subset of these entities



Certification Path

- An entity B requiring to use a public key of an entity A generally needs to obtain and validate a certificate containing the required public key
- This in turn requires that:
 - **the user B knows the public key of the entity C_1 , that signed the certificate (that associates the identity of A to the A 's pub key)**
 - **the user B trusts C_1 ,**
- If the user does not already hold an assured copy of the public key of C_1 , then he might use a certificate of C_1 , as proof of the C_1 's public key
 - **the additional certificate should be signed by a second (trusted) entity C_2 for whom B already holds an assured copy of the public key**
 - **hence, B needs to obtain two certificates: $\text{cert}_{C_2}(C_1)$, $\text{cert}_{C_1}(A)$**
 - **B uses the pub key of C_2 for verifying the cert of C_1 ; then it uses the pub key of C_1 , for verifying the cert of A**



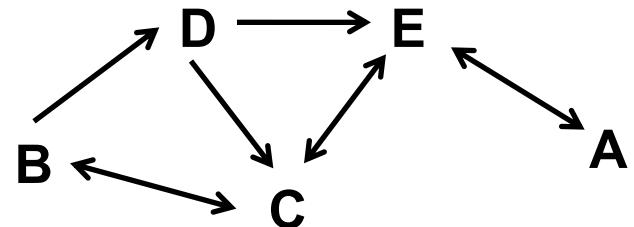
Certification Path (cont.)

- In general, a chain of certificates may be used
 - comprising a certificate of A signed by one entity C_1 , and zero or more additional certificates of C_i (with $i=1,2,\dots$) signed by other entities C_2, C_3, \dots
 - $\text{cert}_{C_n}(C_{n-1}), \text{cert}_{C_{n-1}}(C_{n-2}), \dots, \text{cert}_{C_2}(C_1), \text{cert}_{C_1}(A)$
- If B receives a cert chain, for verifying the cert of A (and all intermediate certs) he just needs the public key of the first entity C_n of the chain, or of any other C_i with $i < n$

Certification Path (cont.)

- Consider the graph G defined as follows:

- the vertexes are the entities
- the edges represent key signatures (certificates)
 - edge from V_1 to V_2 means that V_1 signed the pub key of V_2



- The A's pub key can be securely distributed to B only if
 - B knows the public key of an entity X such that there exists in G a path from X to A
 - the entity B obtains the cert chain from X to A (in order to verify the A's cert)

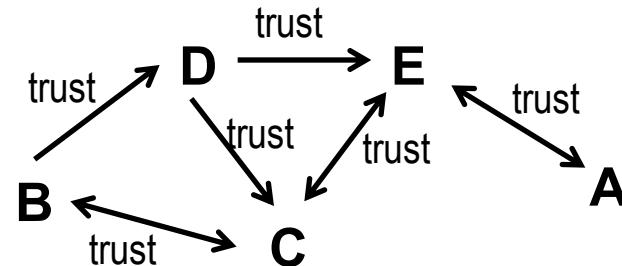


Trust Path

- However, it is not sufficient to obtain the cert path
 - ***B needs also to trust all intermediate entities that signed the certificates included in the cert path***
 - B should trust any C_i (with $i=1,2,\dots,n$) to do correct use of its signature
 - i.e. signing correct certificates (connection between K^+ and its owner)
- Trust delegation (hypothesis)
 - ***In case B doesn't directly know C_i , if B trusts C_{i+1} , and C_{i+1} trusts C_i , then B may decide to trust C_i too***
- If this is done for each C_i (with $i=1,2,\dots,n$), a trust relationship from B to A (or to C_1) can be built based on the trust of B in C_n , and on the trust of C_i in C_{i-1} with $i=2,3,\dots,n$.
 - ***B trusts C_n that trusts C_{n-1} that trusts C_{n-2}, \dots , that trusts C_2 that trust C_1 (that possibly trust A)***
- Trust path
$$B \xrightarrow{\text{trust}} C_n \xrightarrow{\text{trust}} C_{n-1} \xrightarrow{\text{trust}} \dots \xrightarrow{\text{trust}} C_2 \xrightarrow{\text{trust}} C_1 \xrightarrow{\text{(trust)}} A$$

Trust Path (cont.)

- In some certificate applications, an entity signs a certificate only when he/she also trusts the certificate owner
 - **in this case, a cert path leads to a trust path**
 - the cert graph leads to a network of trust relationships
 - in some certificate applications the trust delegation is automatic (from the certificate), in some other cases it is left to the user to decide whether to trust or not a signer/certificate



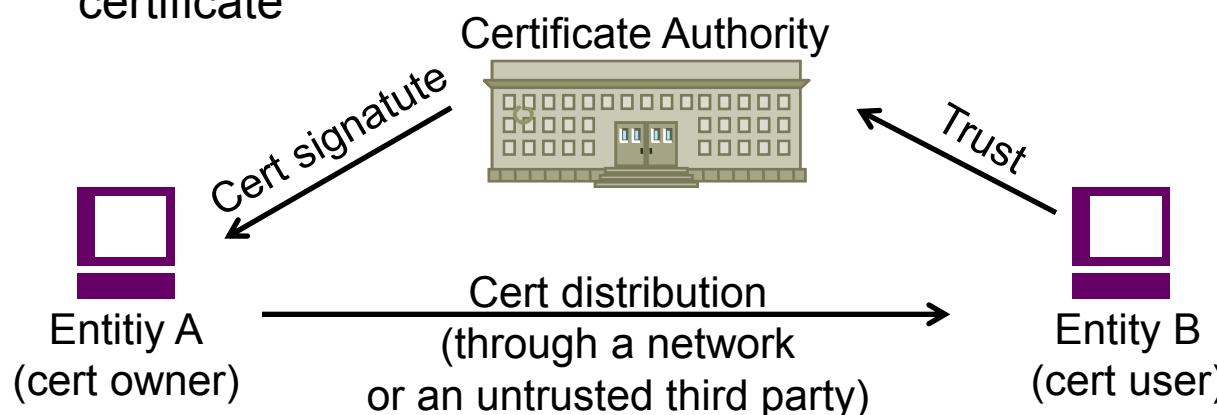


Ceritification Autorities

- In some cases, only specific (trusted) entities are allowed to sign digital certificates
 - **Certification Auhtorities (CAs)**

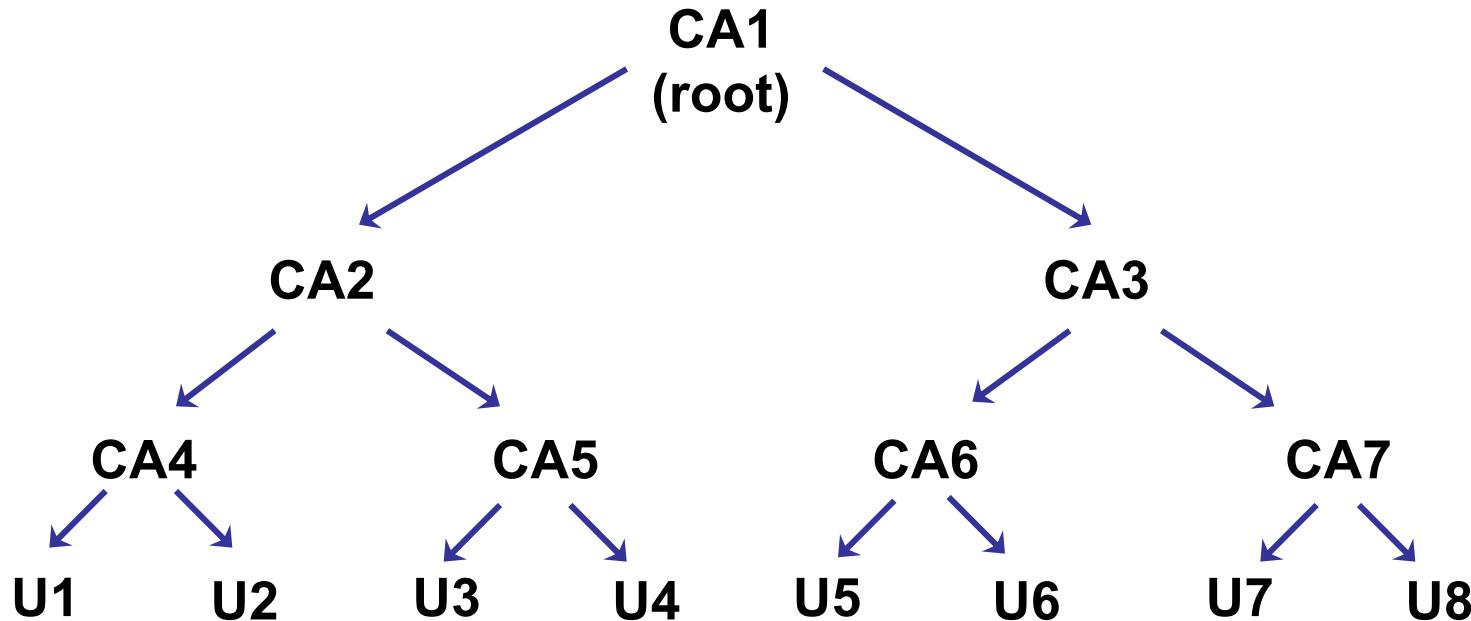
Certificate Authority

- A Certificate Authority (CA) is a trusted third party entity that issues digital certificates
 - **acts as a third party entity for certifying public keys**
 - sometimes it is the only entity entitled to do it
 - **guarantees the “connection” between public keys and their owners**
 - **trusted by the party relying upon the certificate (cert user)**
 - if a user trusts the CA and can verify the CA's signature, then she/he can also assume that a certain public key included in a certificate does indeed belong to whoever is identified in the certificate



Certificate Authority (cont.)

- CAs can be certified by other upper level CAs
- CAs may be organized in a hierarchical (trust) structure





Types of Certificates

- Depending on who signed the certificate:
 - **CA-signed certificate**
 - the certificate signature is provided by a CA
 - **User-signed certificate**
 - the certificate signature is provided by an other user
 - Note: not in X.509 standard
 - **Self-signed certificate**
 - the certificate signature is provided by the owner of the certificate



Types of Certificates (cont.)

- Depending on the owner of the certificate:

- **End system (or User) certificates**

- for binding a user's public key to its owner
 - e.g.
 - User Certificates
 - » for email or other uses
 - Server Certificates
 - » for use by SSL/TLS servers
 - Software Signing Certificates
 - » for signing executable code

- **CA Certificates**

- for verifying signatures on issued (user or CA) certificates

- **Root Certificates**

- self-signed by a CA



Trust models

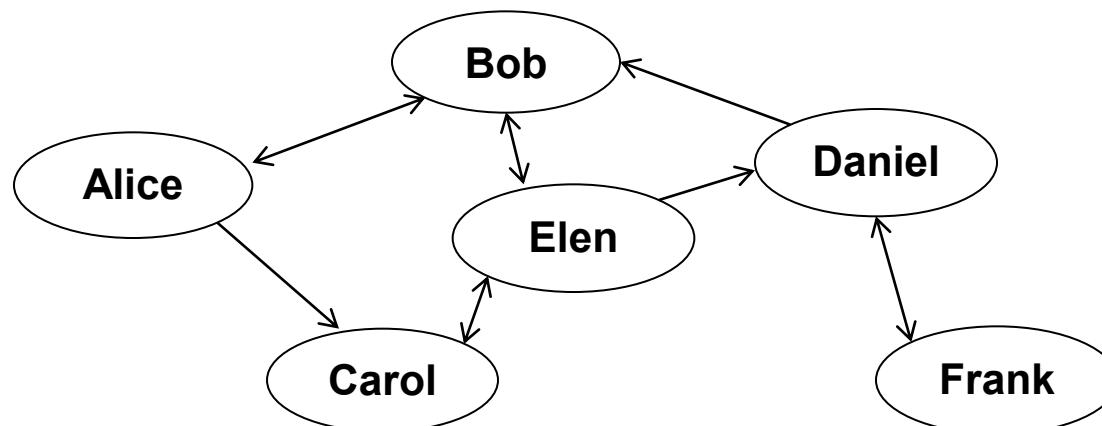
● Public Key Infrastructure (PKI)

- **public keys are signed only by CAs**
 - there are only CA signed certificates (and self signed CA certificates)
- **PKI is a system for the creation, storage, and distribution of digital certificates, based on CAs (and on RAs and VAs)**
- **it consists of:**
 - at least one Certification Authority (CA), which issues and revokes certificates
 - a registration authority (RA) that acts as the verifier for the certificate authority before a digital certificate is issued to a requestor
 - a Validation Authority (VA) that can be used to validate certificates
 - e.g. publicly accessible repository (directory service) which stores Certificate Revocation Lists (CRL)
- **centralized trust model**
- **more CAs may be organized in a hierarchical (trust) structure**

Trust models (cont.)

- Web of Trust (WoT)

- decentralized trust model
- users keys are digitally signed by other users
 - i.e. user-signed certificates are used
 - the same key may be signed/certified by more than one user
 - certifying signatures
 - trust decision is left in the hands of individual users that receives such certificates





Digital certificate standards

- Widely used standards for digital certificates are X.509 and PGP
 - **ITU-T X.509 is the most common standard for digital certificates**
 - uses a PKI, i.e. hierarchical CAs
 - **PGP (Pretty Good Privacy) is mainly used for email**
 - It uses a web of trust

X.509 Digital Certificates



X.509

- ITU-T standard for a Public Key Infrastructure
- X.509 specifies, amongst other things, standard formats for:
 - **public key certificates**
 - **certificate revocation lists**
 - **attributes, and**
 - **a certification path validation algorithm**
- Some X.509 standards:
 - **IETF, RFC 5280: “Internet X.509 Public Key Infrastructure Certificate and CRL Profile”**
 - describes the X.509 v3 certificate and X.509 v2 Certificate Revocation List (CRL) for use in the Internet
 - **IETF, RFC 3647: “Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework”**
 - describes a framework to assist the writers of certificate policies or certification practice statements



X.509 History

- ITU-T X.509 (formerly CCITT X.509) or ISO/IEC/ITU 9594-8, which was first published in 1988 as part of the X.500 Directory recommendations, defines a standard certificate format
 - The certificate format in the 1988 standard is called the version 1 (v1) format
 - The Internet Privacy Enhanced Mail (PEM) RFCs, published in 1993, include specifications for a PKI based on X.509 v1 certificates
 - When X.500 was revised in 1993, two more fields were added, resulting in the version 2 (v2) format
 - The experience gained in attempts to deploy PEM RFCs made it clear that the v1 and v2 certificate formats are deficient in several respects
- ISO/IEC/ITU and ANSI X9 developed the X.509 certificate format
 - first version in June 1996
 - current version 3, 2008 (RFC 5280)
 - extends the v2 format by adding provision for additional extension fields



X.509 (cont.)

- The power of the X.509 model comes from its default arrangement of delegating the trust decision to CAs
- It does this by assuming trust is inherited from the signing key
- Vendors of X.509 products generally include a set of root certificates that the product will trust “out of the box”
 - **therefore automatically validate other certificates presented to the product**
 - **example**
 - X.509 encryption and signature capabilities are built into many web browsers and mail programs
 - the secure HTTP protocol (HTTPS) uses X.509

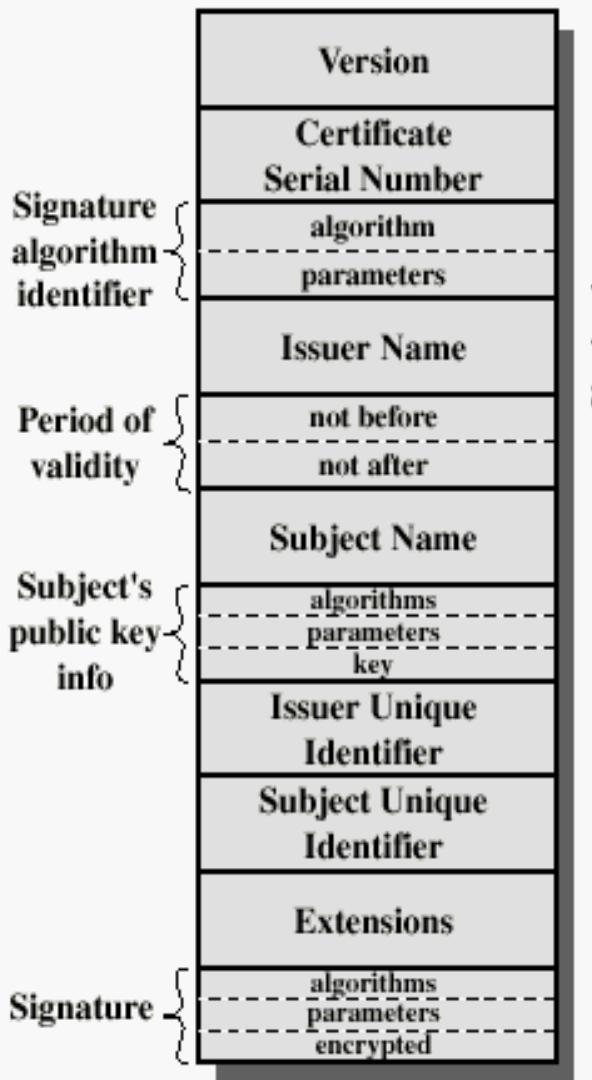


X.509 Digital Certificate

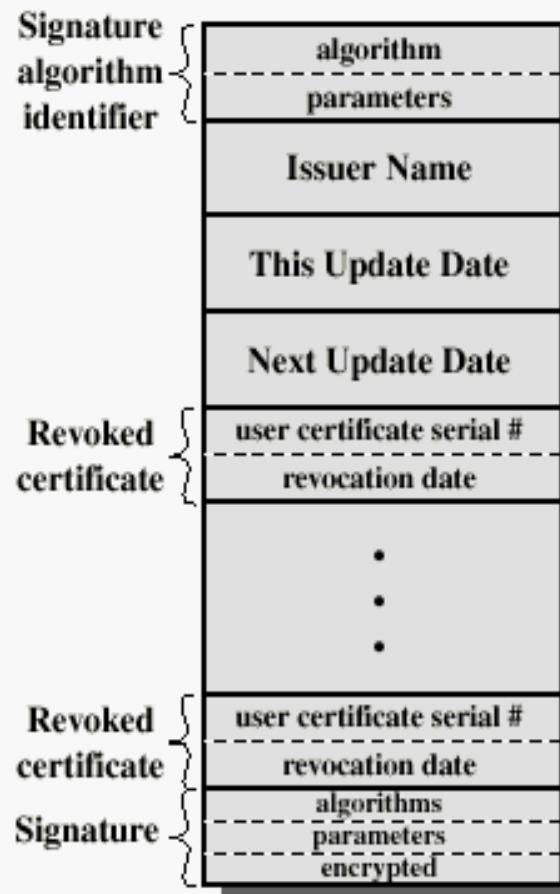
- A X.509 digital certificate includes:
 - **a public key and key info**
 - **the Distinguished Name and other information of the certificate owner that can be a person (name, e-mail, company, phone number) or system (IP address, etc.)**
 - **the Distinguished Name and other information of the certificate issuer (name, e-mail, phone number)**
 - **version number**
 - **a serial number uniquely associated to the certificate**
 - **the level of trust associated to the certificate**
 - **issue date**
 - **expiration date**
 - **digital signature of the issuer**
 - algorithm
 - signature
 - **extensions (optional)**



X.509 v1,v2,v3 and CRL Formats



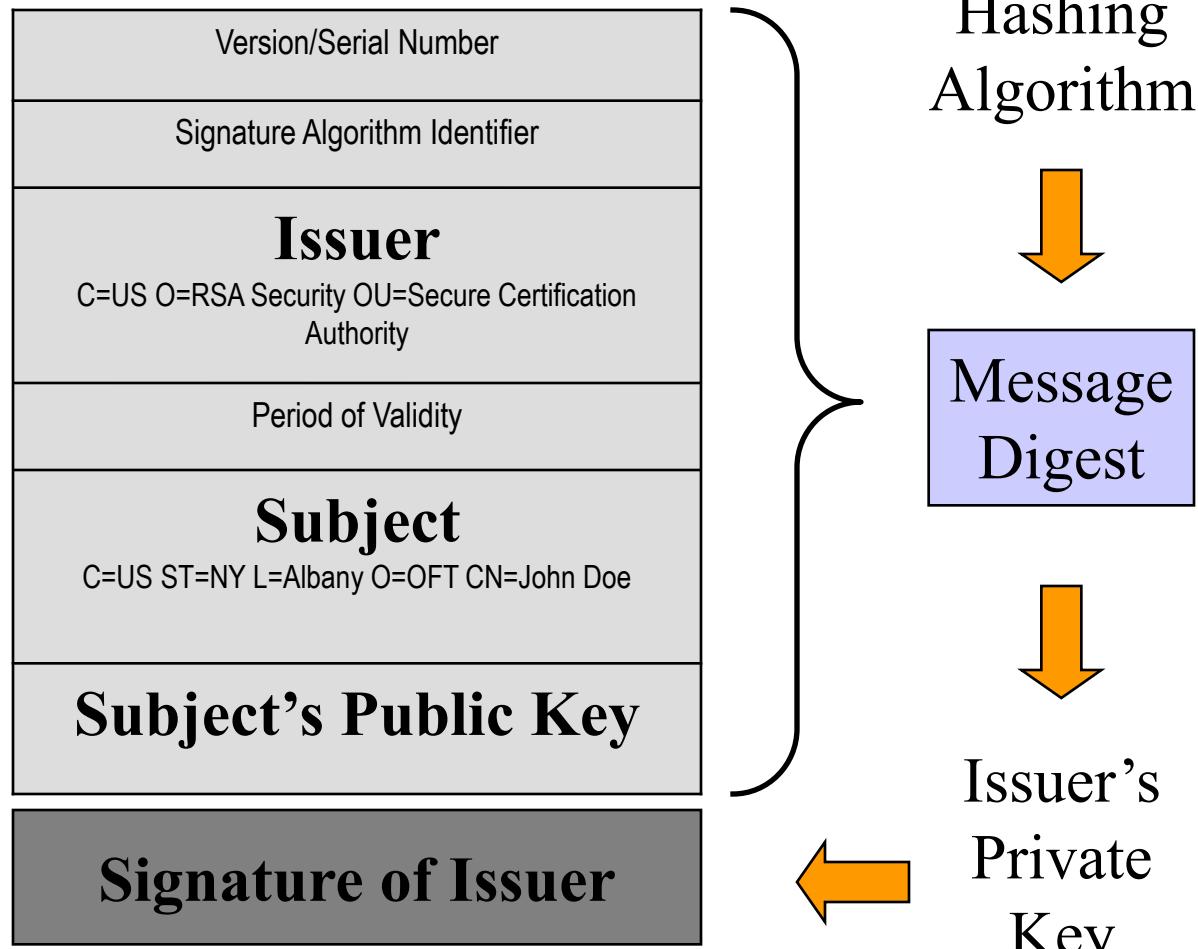
(a) X.509 Certificate



(b) Certificate Revocation List



X.509 Certificate Signature





ASN.1, BER and DER

- ASN.1 (Abstract Syntax Notation One, defined in X.208) is the OSI's method of specifying abstract objects
- ASN.1 is a flexible notation that allows one to define a variety data types
 - **from simple types such as integers and bit strings to structured types such as sets and sequences, as well as complex types defined in terms of others**
- One set of rules for representing such objects as strings of bits is Distinguished Encoding Rules (DER), gives a unique encoding to each ASN.1 value
 - **DER is a subset of BER (Basic Encoding Rules) that describes how to represent or encode values of each ASN.1 type as a string of eight-bit octets**
 - there is generally more than one way to BER-encode a given value, while there is only one DER encoding



Example of PEM-encoded certificate

BEGIN CERTIFICATE

```
MIIC7jCCAl...  
FTATB...  
A1UECh...  
cm10eTEVMBMGA1UEAxMMU25ha2UgT21sIENBMR4wHAYJKoZIhvcNAQkBFg9jYUBz  
bmFrZW9pbC5kb20wHhcNOTgxMDIxMDg1ODM2WhcNOTkxMDIxMDg1ODM2WjCBpzEL  
MAkGA1UEBhMCWFkxF...  
a2UgVG93bjEXMBUGA1UEChMOU25ha2UgT21sLCBMdGQxFzAVB...  
cnZlciBUZWftMRkwFwYDVQQDEXB3d3cuc25ha2VvaWwuZG9tMR8wHQYJKoZIhvcN  
AQkB...  
gQDH9Ge/s2zch+da+rPTx/DPRp3xGjHZ4GG6pCmvADIEtBtKBFAcZ64n+Dy7Np8b  
vKR+yy5DGQiij...  
1WoANFlAz1SdbxeGVHoT0K+gT5w3UxwZKv2DLbCTzLZyPwIDAQAB...  
HRMECDAGAQH/AgEAMBE...  
gQAZUIHAL4D09oE6Lv2k56Gp38OB...  
2q2QoyulCgSzHbEGmi0EsdkPfg6mp0...  
dUHzICxBVC1lnHyYGjDuAMhe3961YAn8bCld1/L4NMGBCQ==
```

END CERTIFICATE



Public-key Cryptography Standard (PKCS)

- A group of public-key cryptography standards, originally developed by RSADSI (RSA Data Security Inc.)
 - **describes the syntax for messages in an abstract manner**
 - **gives complete details about algorithms**
 - **defines encoding for**
 - RSA public/private key,
 - signature,
 - short RSA-encrypted message (typically a secret key),
 - etc
- Some of these standards have moved into the "standards-track" processes of the IETF
- The standards are based on ASN.1 and BER/DER to describe and represent data



PKCS (cont.)

- PKCS #1: RSA Cryptography Standard (RFC 3447)
 - **provides recommendations for the implementation of public-key cryptography based on the RSA, covering the following aspects:**
 - Cryptographic primitives
 - Encryption schemes
 - Signature schemes
 - ASN.1 syntax for representing keys and for identifying the schemes
- PKCS #3: Diffie-Hellman Key Agreement Standard
 - **defines the Diffie–Hellman Key Agreement protocol for establishing a shared secret key between two parties**
- PKCS #5: Password-Based Cryptography Standard (RFC 2898)
 - **provides recommendations for the implementation of password-based cryptography, covering:**
 - key derivation functions
 - encryption schemes
 - message-authentication schemes



PKCS (cont.)

- PKCS #6: Extended-Certificate Syntax Standard
 - **defines extensions to the old v1 X.509 certificate specification**
 - **it has been obsoleted by the X.509v3 standard**
- PKCS #7: Cryptographic Message Syntax Standard (RFC 2315)
 - **describes a general syntax for data that may have cryptography applied to it, such as digital signatures and digital envelopes**
 - **it is compatible with PEM**
 - signed-data and signed-and-enveloped-data content can be converted into PEM messages, and vice versa
- PKCS #8: Private-Key Information Syntax Standard (RFC 5208)
 - **describes a syntax for private-key information (private key and a set of attributes)**
 - **also describes a syntax for encrypted private keys**
 - a password-based encryption algorithm could be used (e.g., one of those described in PKCS#5)



PKCS (cont.)

- PKCS #9: Selected Attribute Types (RFC 2985)
 - **defines selected attribute types for use in**
 - PKCS #6 extended certificates
 - PKCS #7 digitally signed messages
 - PKCS #8 private-key information, and
 - PKCS #10 certificate-signing requests
- PKCS #10: Certification Request Syntax Standard (RFC 2986)
 - **defines a format of messages sent to a certification authority to request certification of a public key**
- PKCS #11: Cryptographic Token Interface Standard
 - **An API defining a generic interface to cryptographic tokens**
 - **Often used in single sign-on, Public-key cryptography and disk encryption systems**
- PKCS #12: Personal Information Exchange Syntax Standard
 - **defines a file format commonly used to store private keys with accompanying public key certificates, protected with a password-based symmetric key**



Certificate Revocation

- When a certificate is issued, it is expected to be in use for its entire validity period (certificates have a period of validity)
- However, various circumstances may cause a certificate to become invalid (revoked) prior to the expiration of the validity period, e.g.
 - **change of name, change of association between subject and CA (e.g., an employee terminates employment with an organization)**
 - **user's private key is assumed (or suspected) to be compromised**
 - **user is no longer certified by this CA**
 - **CA's certificate is assumed to be compromised**
- X.509 defines one method of certificate revocation
- This method involves each CA periodically issuing a signed data structure called a certificate revocation list (CRL)
- Users should check certs with CA's CRL



Certificate Revocation List (CRL)

- A CRL is a time stamped list identifying revoked certificates which is signed by a CA and made freely available in a public repository
- When a system uses a certificate, that system not only checks the certificate signature and validity but also acquires a suitably-recent CRL and checks that the certificate serial number is not on that CRL
- A CA issues a new CRL on a regular periodic basis (e.g., hourly, daily, or weekly)
- An entry is added to the CRL as part of the next update following notification of revocation
 - **an entry may be removed from the CRL after appearing on one regularly scheduled CRL issued beyond the revoked certificate's validity period**
- CRLs may be distributed by exactly the same means as certificates themselves, namely, via untrusted communications and server systems
- One limitation of the CRL revocation method, is that the time granularity of revocation is limited to the CRL issue period



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Secure Communications: IPSec and TLS

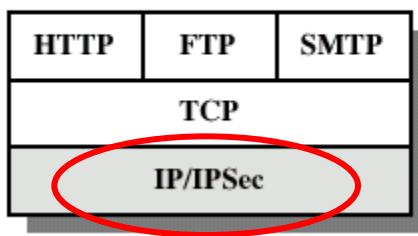
Luca Veltri

(mail.to: luca.veltri@unipr.it)

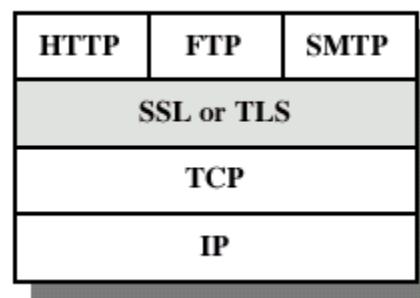
Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>

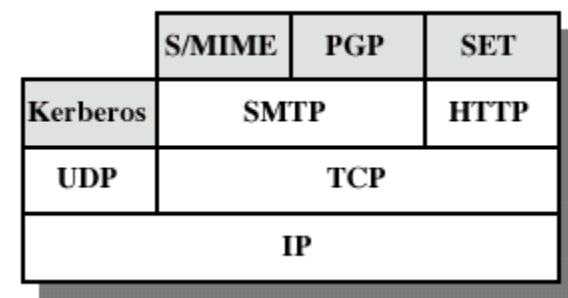
Security within IP stack protocols



Network level



transport/session level



Application level

IPSec



IPSec Introduction

- IPSec, as defined in RFC 4301 and successive standards, is a framework for providing interoperable, high quality, cryptographically-based security for IPv4 and IPv6
 - **offering protection in a standard fashion for all protocols that may be carried over IP (including IP itself)**
- The set of security services offered includes
 - **access control**
 - **connectionless data integrity, data origin authentication**
 - **detection and rejection of replays**
 - **confidentiality**
 - **and limited traffic flow confidentiality**

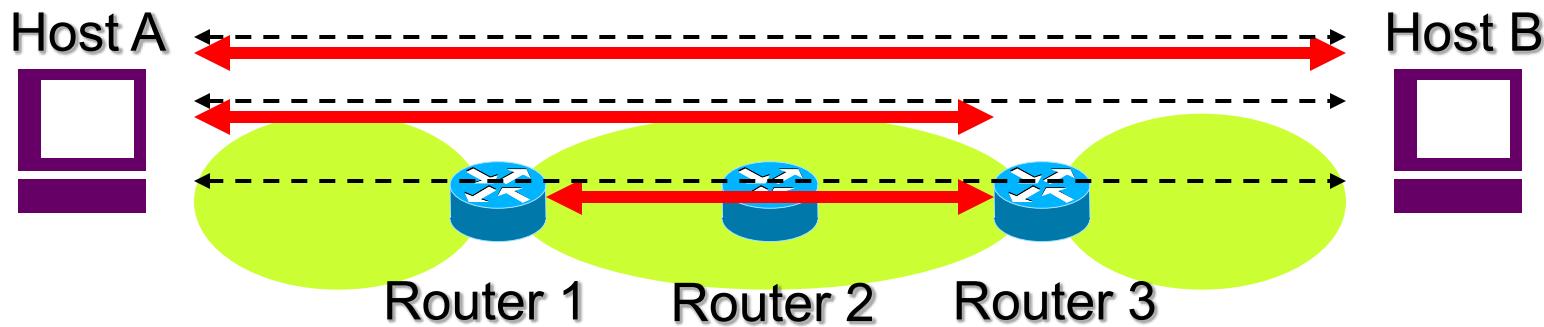


IPSec Introduction (cont.)

- An IPSec implementation operates in a host, or as a security gateway (SG), affording protection to IP traffic
 - **a security gateway is an intermediate system implementing IPSec, e.g., a firewall or router that has been IPSec-enabled**
- IPSec does not adversely affect users, hosts, and other Internet components that do not employ IPSec for traffic protection
 - **transparent for intermediate nodes that do not implement it**
 - **transparent for end nodes (host) that do not implement it, when implemented by intermediate nodes**
 - **transparent for end applications that are not aware of it**
- Hence, IPSec is very useful in presence of legacy hosts or applications
- One IPSec implementation may protect traffic from/to several hosts and applications

IPSec Introduction (cont.)

- IPsec can be used to protect one or more "paths"
 - **between a pair of hosts (host-to-host)**
 - end-to-end, alternative to transport or application level security solutions
 - **between a pair of security gateways (router-to-router)**
 - often referred to as VPN/IPSec
 - **between a security gateway and a host (host-to-router)**
 - often referred to as “road-warrior”



Transport & Tunnel Modes

- IPSec supports two modes of use:

- **transport mode**

- IPSec provide protection primarily next layer protocols
 - The source and destination are the IPSec endpoints



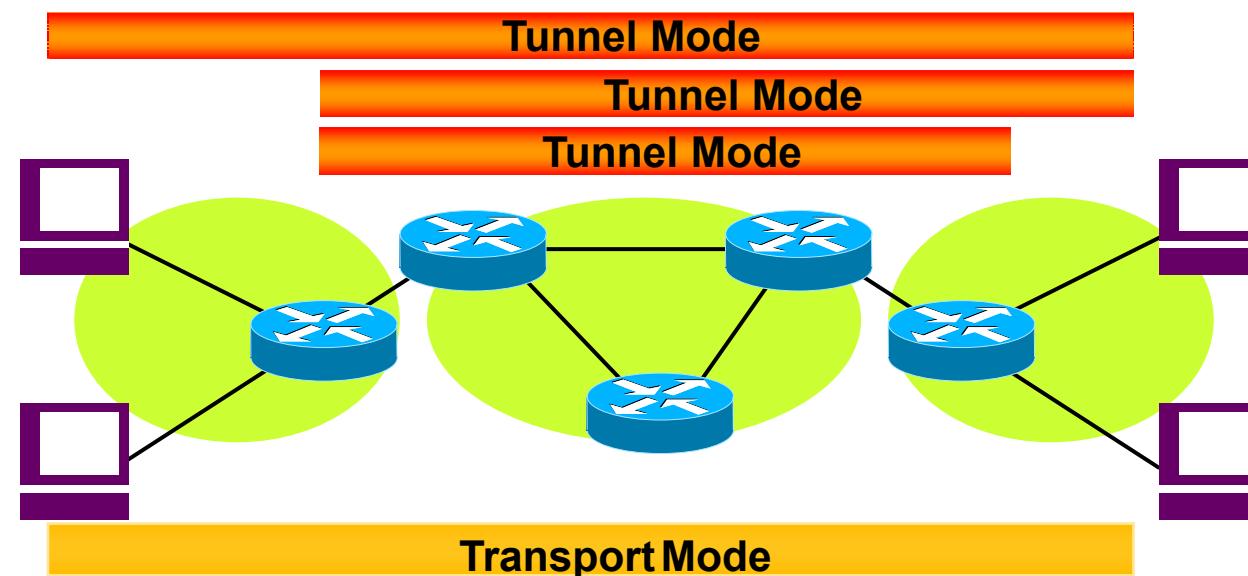
- **tunnel mode**

- IPSec is applied to tunneled IP packets
 - The outer IP header Source and Destination Addresses identify the "endpoints" of the tunnel (the encapsulator and decapsulator)
 - The inner IP header Source and Destination Addresses identify the original sender and recipient of the datagram
 - The inner IP header is not changed except for TTL and the DS fields



Transport & Tunnel Modes (cont.)

- Transport mode can be used only when source and destination addresses coincide with the addresses of the IPSec endpoints





IPSec protocols

- Two protocols have been originally specified for securing IPSec communications
 - **the Authentication Header (AH)**
 - originally designed to offer integrity and data origin authentication without confidentiality, with optional (at the discretion of the receiver) anti-replay features
 - support is optional
 - rarely used
 - **Encapsulating Security Payload (ESP)**
 - offers integrity, data origin authentication, and data confidentiality
 - there are also provisions for limited traffic flow confidentiality, i.e., provisions for concealing packet length, and for facilitating efficient generation and discard of dummy packets
 - this capability is likely to be effective primarily in virtual private network (VPN) and overlay network contexts



IPSec protocols (cont.)

- support for AH is optional and experience has shown that there are very few contexts in which ESP cannot provide the requisite security services
 - **ESP can be also used to provide only integrity, without confidentiality, making it comparable to AH in most contexts**
 - **as a result, AH is rarely used**
- Because IPSec security services require the use of cryptographic keys, IPSec relies on a separate mechanism for putting these keys in place
 - **IPSec specifies IKE (Internet Key Exchange) as public-key based key management mechanism**
 - **other automated key distribution techniques may also be used**

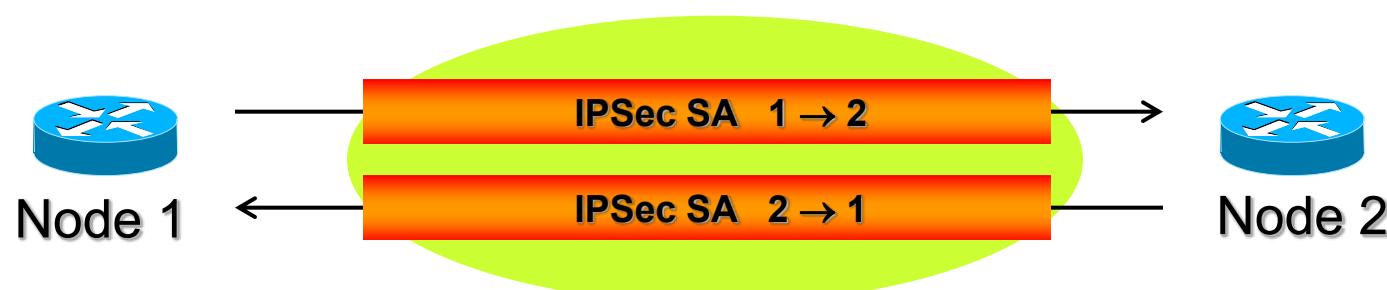


Cryptographic algorithm independency

- IPsec security protocols (ESP, AH, and IKE) are designed to be cryptographic algorithm independent
 - **this modularity permits selection of different sets of cryptographic algorithms as appropriate, without affecting the other parts of the implementation**
 - **different users may select different sets of cryptographic algorithms**
 - **to facilitate interoperability in the global Internet, a set of default cryptographic algorithms for use with AH and ESP, and a set of mandatory-to-implement algorithms for IKEv2, are specified**

Security Association (SA)

- A Security Association (SA) is a simplex logical "connection" that affords security services to the traffic carried by it
 - specifies algorithms, key material, and parameters to operate the AH and/or ESP operations
 - a major function of IKE is the establishment and maintenance of SAs
 - to secure typical, bi-directional communication between two IPsec-enabled systems, a pair of SAs (one in each direction) is required





Encapsulating Security Payload (ESP)

- Encapsulating Security Payload (ESP) (RFC 4303) is designed to provide a mix of security services
 - **confidentiality**
 - **data origin authentication and connectionless integrity**
 - **anti-replay service (a form of partial sequence integrity)**
 - **(limited) traffic flow confidentiality**
 - by adding padding bytes after the end of the payload data
- Confidentiality and integrity can be offered independently
 - **only data authentication**
 - ESP is simpler and faster to process than AH
 - **only encryption**
 - this will provide defense only against passive attackers
 - using encryption without a strong integrity mechanism may render the confidentiality service insecure against some forms of active attacks
- ESP typically will employ both services



ESP (cont.)

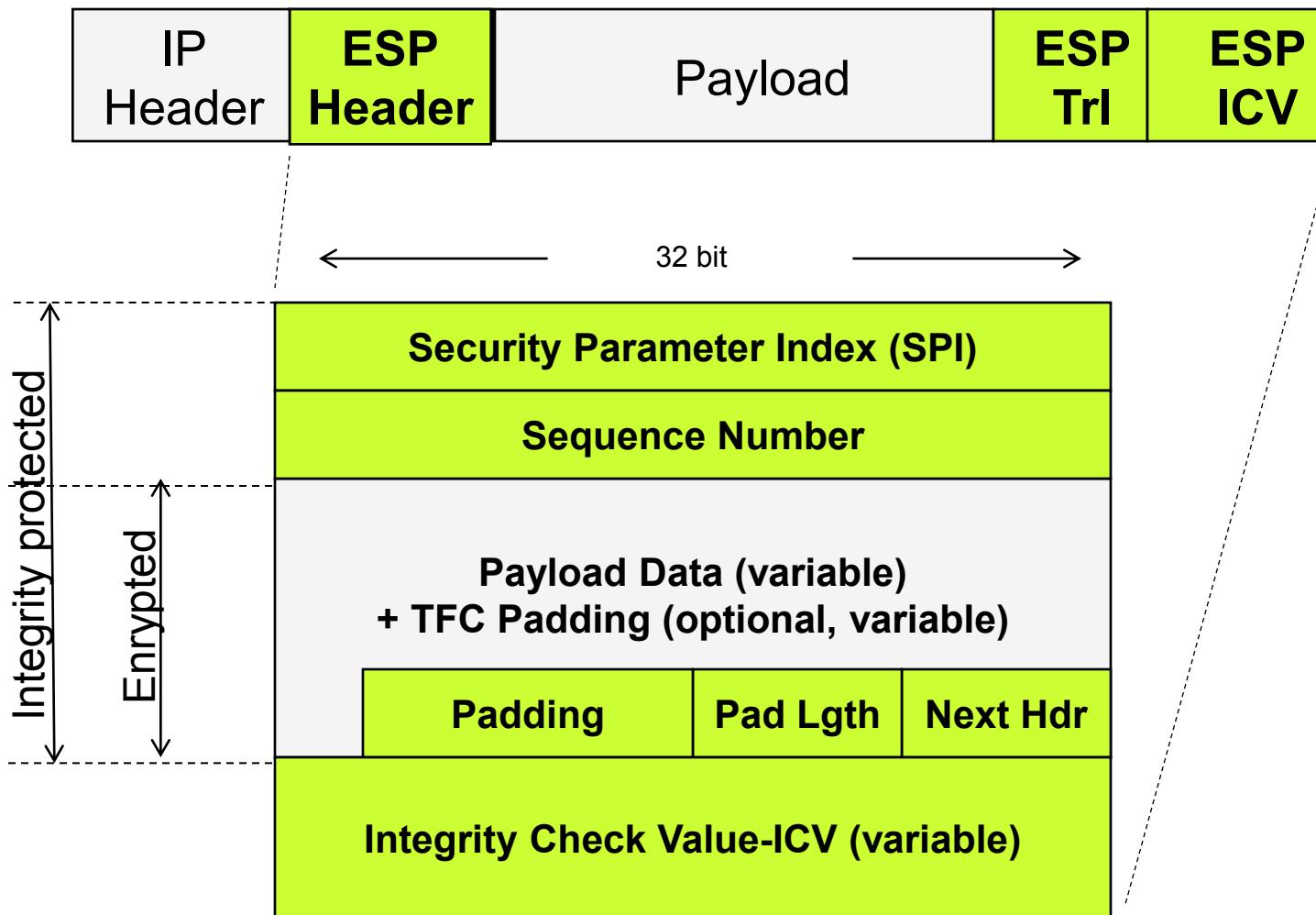
- The anti-replay service may be used only if the integrity service is provided
 - **the selection of this service is solely at the discretion of the receiver and thus need not be negotiated**
- The traffic flow confidentiality (TFC) service generally is effective only if
 - **ESP is employed in a fashion that conceals the ultimate source and destination addresses of correspondents, e.g., in tunnel mode between SEGs, and**
 - **sufficient traffic flows between IPsec peers (either naturally or as a result of generation of masking traffic) to conceal the characteristics of specific, individual subscriber traffic flows**



ESP (cont.)

- ESP may be applied alone, in combination with AH, or in a nested fashion
- The ESP header is inserted after the IP header and before the next layer protocol header (transport mode) or before an encapsulated IP header (tunnel mode)

ESP Packet Format





ESP Packet Format (cont.)

- Security Parameter Index (SPI)
 - **an arbitrary 32-bit value that is used by a receiver to identify the SA to which an incoming packet is bound**
- Sequence number
 - **unsigned 32-bit field contains a counter value that increases by one for each packet sent, i.e., a per-SA packet sequence number**
- Payload data
 - **variable-length field containing data (from the original IP packet) described by the Next Header field**
 - If the algorithm used to encrypt the payload requires cryptographic synchronization data, e.g., an Initialization Vector (IV), then this data is carried explicitly in the Payload field
- Traffic Flow Confidentiality (TFC) Padding
 - **added only if the Payload Data field contains a specification of the length of the IP datagram**
 - this is always true in tunnel mode, and may be true in transport mode depending on whether the next layer protocol (e.g., IP, UDP, ICMP) contains explicit length information



ESP Packet Format (cont.)

- Padding (for Encryption)
 - **0 to 255 bytes of padding**
 - If the encryption algorithm requires the plaintext to be a multiple of some number of bytes, e.g., the block size of a block cipher
 - or to ensure that the resulting ciphertext terminates on a 4-byte boundary
 - note: a separate mechanism should be used for TFC (if required)
- Pad length
 - **the number of pad bytes in the Padding field**
- Next header
 - **8-bit field that identifies the type of data contained in the Payload Data field, e.g., an IPv4 or IPv6 packet, or a next layer header and data**
- Integrity Check Value (ICV)
 - **(optional) variable-length field computed over the ESP header, Payload, and ESP trailer fields**

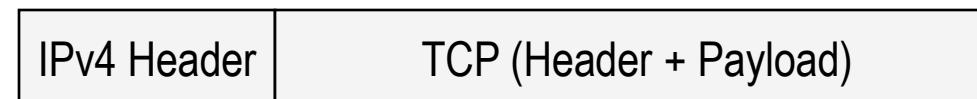


ESP Packet Format (cont.)

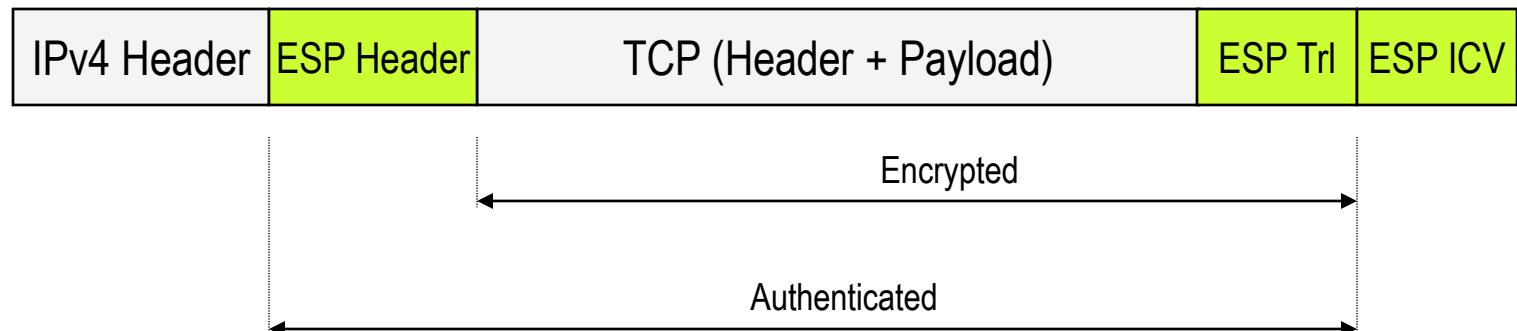
- If the confidentiality service is selected, the encrypted data consists of the Payload Data and the ESP trailer
- If the integrity service is selected, the integrity computation encompasses the SPI, Sequence Number, Payload Data, and the ESP trailer

ESP - Transport Mode

- In transport mode, ESP is inserted after the IP header and before a next layer protocol, e.g., TCP, UDP, ICMP, etc.
 - **in IPv4, this translates to placing ESP after the IP header (and any options that it contains), but before the next layer protocol**
- Example of IPv4 packet before applying ESP

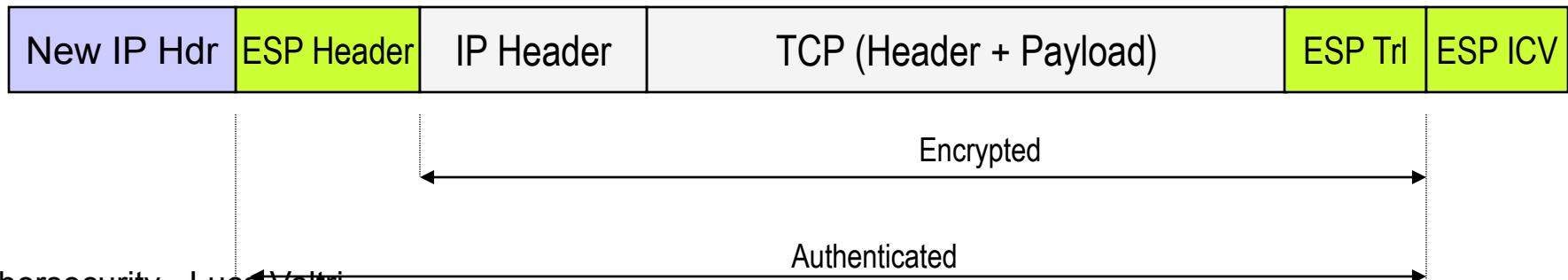
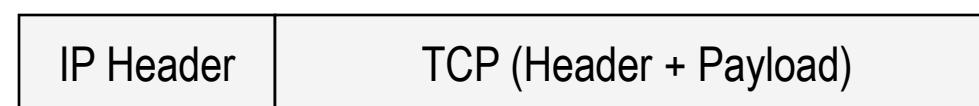


- The same packet after applying ESP in transport mode



ESP - Tunnel Mode

- In tunnel mode, the "inner" IP header carries the ultimate (IP) source and destination addresses, while an "outer" IP header contains the addresses of the IPsec "peers", e.g., addresses of SEGs
 - mixed inner and outer IP versions are allowed, i.e., IPv6 over IPv4 and IPv4 over IPv6
 - ESP protects the entire inner IP packet, including the entire inner IP header

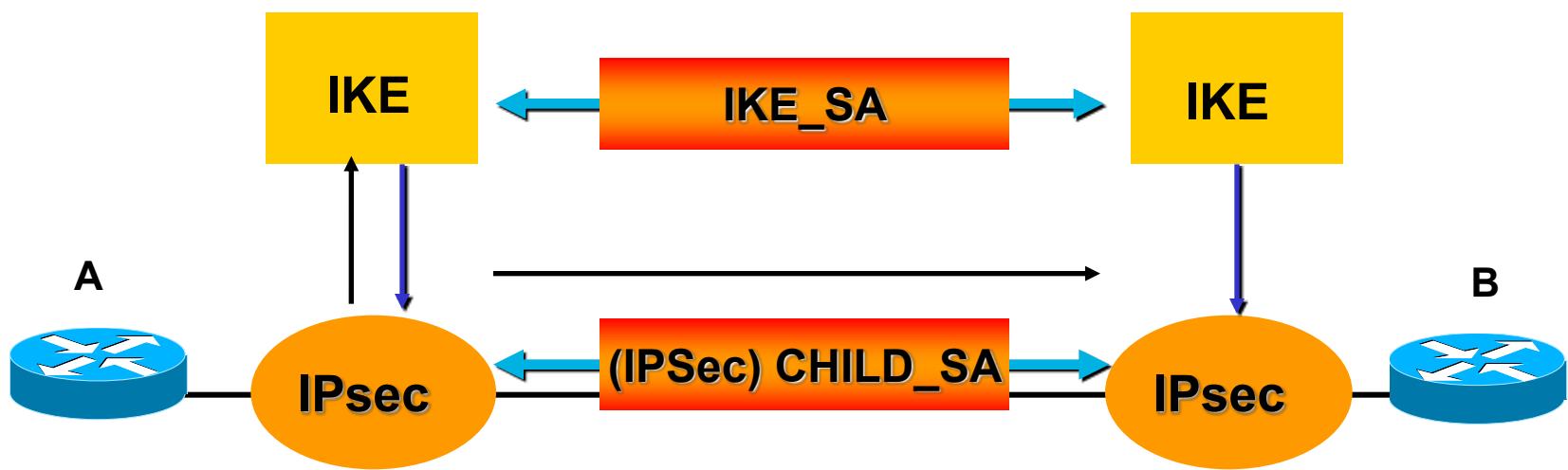




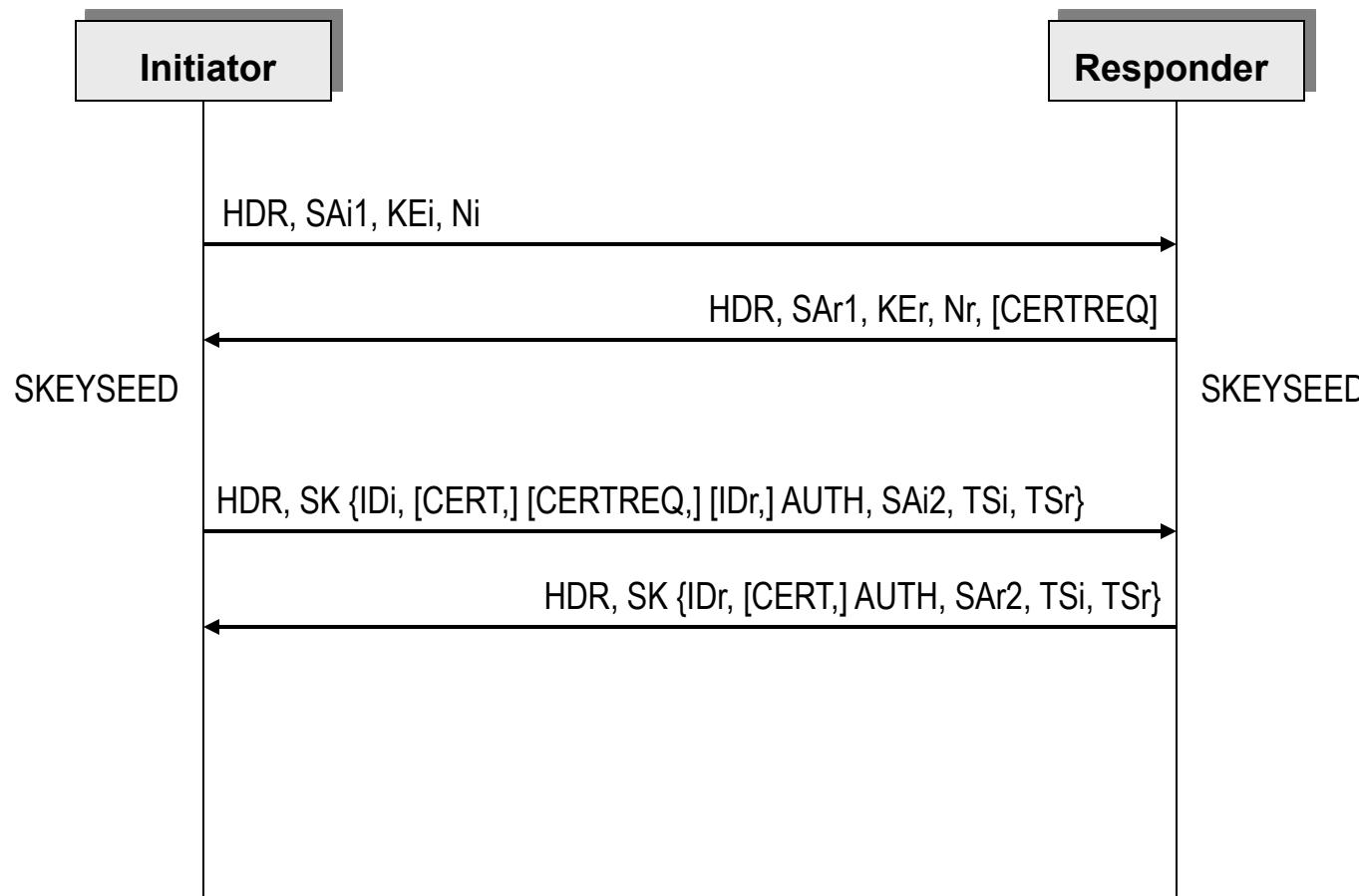
Internet Key Exchange (IKEv2) Protocol

- IP Security (IPSec) provides confidentiality, data integrity, access control, and data source authentication to IP datagrams
- These services are provided by maintaining shared state (the Security Association) between the source and the destination
 - **this state defines the specific services provided to the datagram, which cryptographic algorithms will be used to provide the services, and the keys used as input to the cryptographic algorithms, etc.**
 - **establishing this state in a manual fashion does not scale well**
- Internet Key Exchange (IKEv2) performs mutual authentication between two parties and establishes an IKE SA (IKE_SA) that includes shared secret information that can be used to efficiently establish SAs for ESP and/or AH (CHILD_SAs)

IKE_SA and CHILD_SAs



Initial Exchanges: IKE_SA_INIT and IKE_AUTH





IKE message notation

- Notation:

AUTH	Authentication
CERT	Certificate
CERTREQ	Certificate Request
CP	Configuration
D	Delete
EAP	Extensible Authentication
HDR	IKE header (not a payload)
IDi	Identification - Initiator
IDr	Identification - Responder
KE	Key Exchange
Ni, Nr	Nonce
N	Notify
SA	Security Association
SK	Encrypted and Authenticated
TSi	Traffic Selector - Initiator
TSr	Traffic Selector - Responder
V	Vendor ID



Initial Exchanges: IKE_SA_INIT and IKE_AUTH (cont.)

- IKE always begins with IKE_SA_INIT and IKE_AUTH exchanges (known in IKEv1 as Phase 1)
 - HDR contains the SPIs, version numbers, and flags
 - SA states the cryptographic algorithms for the IKE_SA
 - The KE contains the Diffie-Hellman value
 - Ni and Nr are nonce values
 - **SKEYSEED is the DH secret from which all keys are derived for that IKE_SA**
 - all but the headers of all the messages that follow are encrypted and integrity protected by $SK\{\cdot\}$
 - SKEYSEED is calculated as: $= PRF(Ni \mid Nr, g^{ir})$
 - from SKEYSEED further secrets are then generated:
$$SK_d \mid SK_{ai} \mid SK_{ar} \mid SK_{ei} \mid SK_{er} \mid SK_{pi} \mid SK_{pr} = prf+(SKEYSEED, Ni \mid Nr \mid SPI_i \mid SPI_r)$$
 - SK_e and SK_a are used for IKE encryption and integrity protection
 - SK_d is used for derivation of further keying material for CHILD_SAs
 - SK_p are used in the input data of AUTH



Initial Exchanges: IKE_SA_INIT and IKE_AUTH (cont.)

- The initiator and responder assert their identities with the ID payload
- AUTH payload is the proof of knowledge of the secret corresponding to ID and is used to integrity protect the contents of the IKE_SA_INIT messages
- Optionally, messages 3 and 4 may include a certificate, or certificate chain providing evidence that the key used to compute a digital signature belongs to the name in the ID payload
- The optional payloads IDr (for initiator) and IDi (for responder) enable both entities to specify which peer's identities they wants to talk to
- The optional payloads TSi and TSr report the proposed traffic selectors (in the 3rd message) and the accepted subset of traffic selectors (in the 4th message)
 - IP packet "protect" selectors are stored in a Security Policy Database (SPD)



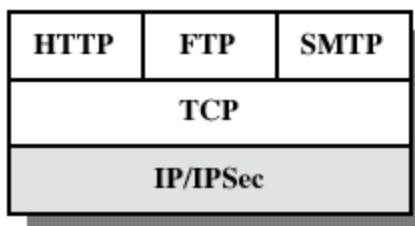
Authentication of the IKE SA

- The signature or MAC will be computed using algorithms dictated by the type of key used by the signer, and specified by the Auth Method field in the Authentication payload
- In case of shared secret
 - **AUTH = prf(prf(Shared Secret,"Key Pad for IKEv2"), <data>)**
 - the pad string is added so that if the shared secret is derived from a password, the IKE implementation can store the value prf(Shared Secret,"Key Pad for IKEv2")
- In case of public-key
 - RSA or DSA digital signature over the hash value
- Data to be “signed” start with the first octet of the first SPI in the header and end with the last octet of the last payload
 - for the responder, appended to this are the initiator's nonce Ni and the value prf(SK_pr, IDr') where IDr' is the responder's ID payload excluding the fixed header
 - for the initiator, appended to this are the responder's nonce Nr, and the value prf(SK_pi, IDi')

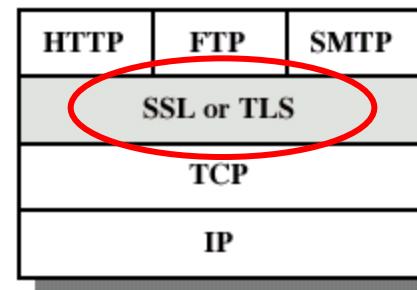
TLS and DTLS

Transport Layer Security (TLS)

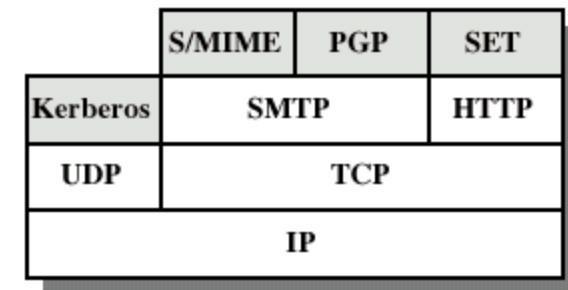
- Transport Layer Security (TLS) protocol (RFC 5246, version 1.2) provides privacy and data integrity between two communicating client/server applications
- TLS is application protocol independent
 - higher-level protocols can layer on top of the TLS protocol transparently



Network level



transport/session level



Application level



TLS security services

- TLS provides:
 - **peer's identity authentication**
 - using asymmetric cryptography (e.g., RSA, DSA, etc.)
 - **negotiation of encryption algorithm and cryptographic keys**
 - protected against eavesdropping, modifying and replay attacks
 - **confidentiality**
 - symmetric cryptography is used
 - **Message integrity and data authentication**
 - message transport includes a message integrity check using a keyed MAC
 - secure hash functions (e.g., SHA-1, etc.) are used for MAC computations
- TLS uses X.509 server certificates and client certificates (optional)



TLS vs. SSL

- TLS protocol (IETF standard) was based on the SSL 3.0 Protocol Specification as published by Netscape
- TLS v1.0 == SSL v3.1
 - **the differences between TLS v1.0 and SSL 3.0 were minor:**
 - record format version number
 - uses HMAC for MAC
 - a pseudo-random function expands secrets
 - has additional alert codes
 - some changes in supported ciphers
 - changes in certificate negotiations
 - changes in use of padding

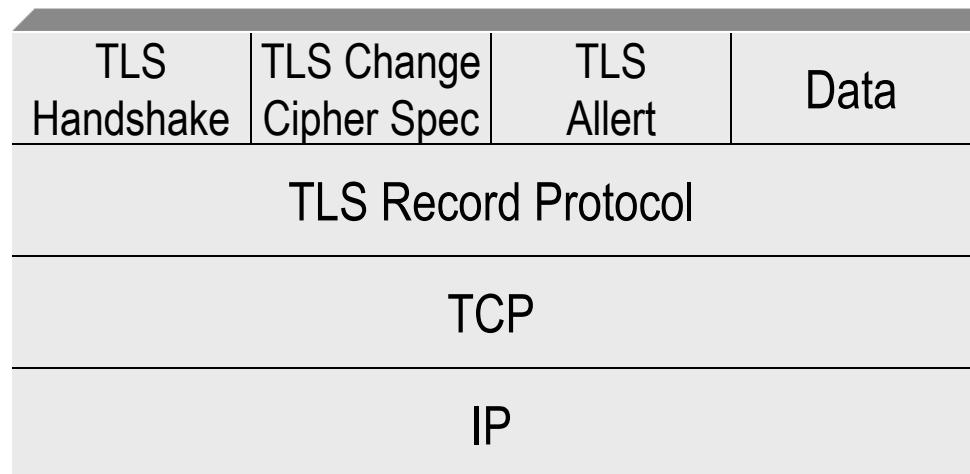


TLS phases

- 1) Handshake
 - Establish connection
 - Agree on encryption algorithm
 - Exchange key
 - Authentication
 - through certificates
 - server only authentication, or both client and server
- 2) Securing messages
 - Sending the actual encrypted messages
 - Integrity checks with MACs

TLS protocol

- The TLS Protocol is a layered protocol
 - At each layer, messages may include fields for length, description, and content
- The protocol is composed of two layers:
 - **TLS Handshake Protocol**
 - **TLS Record Protocol**



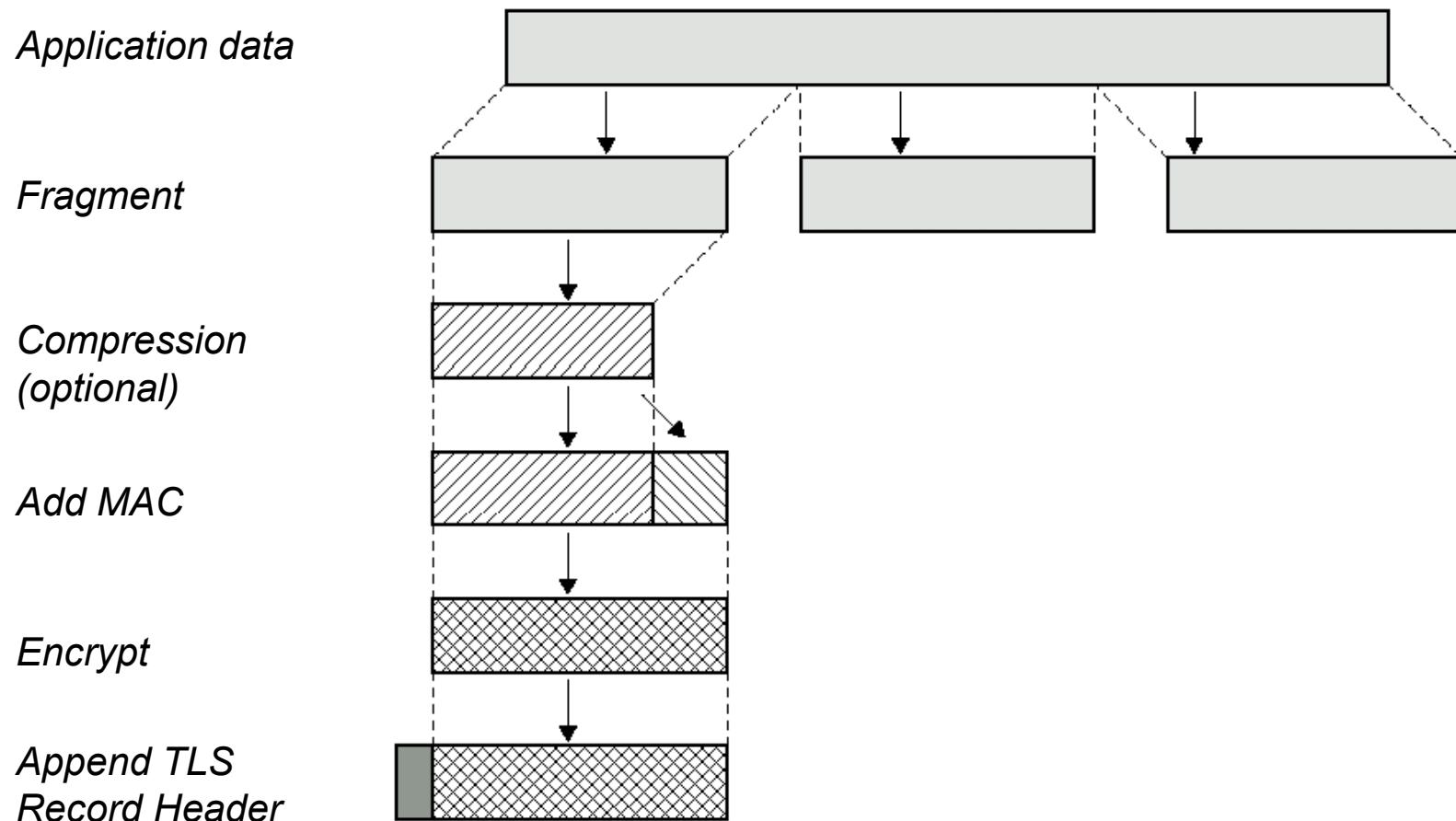


TLS Record Protocol

- TLS Record Protocol is at the lowest level, layered on top of some reliable transport protocol (e.g., TCP)
- Fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the result
- Provides connection security:
 - **confidentiality**
 - symmetric cryptography is used for data encryption (e.g., AES, RC4, etc.)
 - the keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol)
 - **Message integrity and data authentication**
 - message transport includes a message integrity check using a keyed MAC
 - secure hash functions (e.g., SHA-1, etc.) are used for MAC computations
- It is used for encapsulating higher-level TLS protocols
 - **currently, four protocols are considered: the handshake protocol, the alert protocol, the change cipher spec protocol, and the application data protocol**

TLS Record Protocol (cont.)

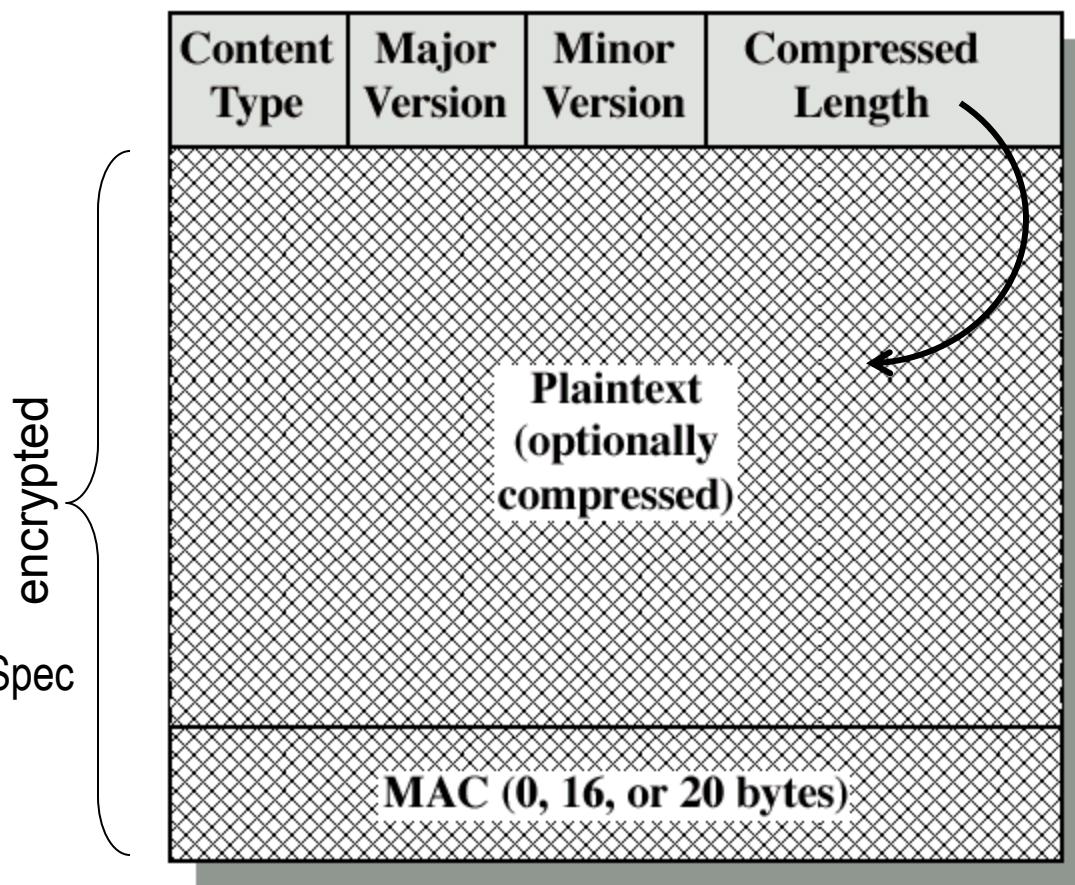
- TLS-RP operation:



TLS Record Protocol (cont.)

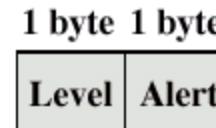
● TLS-RP format

- Major Version=3
- Minor Version:
 - 0 = SSL3.0
 - 1 = TLS1.0
 - 2 = TLS1.1
 - 3 = TLS1.2
 - 4 = TLS1.3
- Content-type:
 - 20 = Change Cipher Spec
 - 21 = Alert
 - 22 = Handshake
 - 23 = Data



TLS Alert Protocol

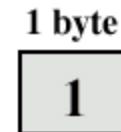
- Alert messages convey the severity of the message (warning or fatal) and a description of the alert
- Message consists of two bytes (Level type and Alert code)



- Two levels of alerts are defined
 - **warning (first byte=1)**
 - close notify, no certificate, bad certificate, unsupported certificate, certificate revoked, certificate expired, certificate unknown
 - **fatal (first byte=2)**
 - unexpected message, bad record mac, decompression failure, handshake failure, illegal parameter
 - result in the immediate termination of the connection
- Like other messages, alert messages are encrypted
- TLS-RP Content-type 21

TLS Change Cipher Spec Protocol

- The change cipher spec protocol exists to signal transitions in ciphering strategies
- The protocol consists of a single message, which is encrypted and compressed under the current (not the pending) connection state
- ChangeCipherSpec message consists of a single byte of value 1



- The message is sent by both the client and the server to notify the receiving party that subsequent records will be protected under the newly negotiated CipherSpec and keys
- TLS-RP Content-type 20



TLS Handshake Protocol

- TLS Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data
- The TLS Handshake Protocol provides:
 - **authentication of the peer's identity using asymmetric cryptography (e.g., RSA, DSA, etc.)**
 - **secure negotiation of a shared secret**
- TLS-RP Content-type 22



Original cert-based key exchanges

- DHE-RSA and ECDHE-RSA
 - DH exchange signed using RSA keys
 - forward secrecy
- DHE_DSS
 - DH exchange signed using DSS keys
 - forward secrecy
- RSA
 - key exchanged encrypted by the client with the RSA key of the server
 - has been removed in TLS 1.3
- DH_RSA, ECDH-RSA, DH_DSS, and ECDH-ECDSA
 - use static DH cert (signed with RSA/DSS)
 - have been removed in TLS 1.3



TLS 1.3 key exchanges

- TLS 1.3 supports three basic key exchange modes:
 - **(EC)DHE**
 - Diffie-Hellman over either finite fields or elliptic curves
 - **PSK-only**
 - **PSK with (EC)DHE**



TLS Handshake Protocol (cont.)

- When a TLS client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets
- The TLS Handshake Protocol involves the following steps:
 - **exchange hello messages to agree on algorithms, exchange random values, and check for session resumption**
 - **exchange the necessary cryptographic parameters to allow the client and server to agree on a pre-master secret**
 - **exchange certificates and cryptographic information to allow the client and server to authenticate themselves**
 - **generate a master secret from the premaster secret and exchanged random values**
 - **provide security parameters to the record layer**
 - **allow the client and server to verify that their peer has calculated the same security parameters and that the handshake occurred without tampering by an attacker**



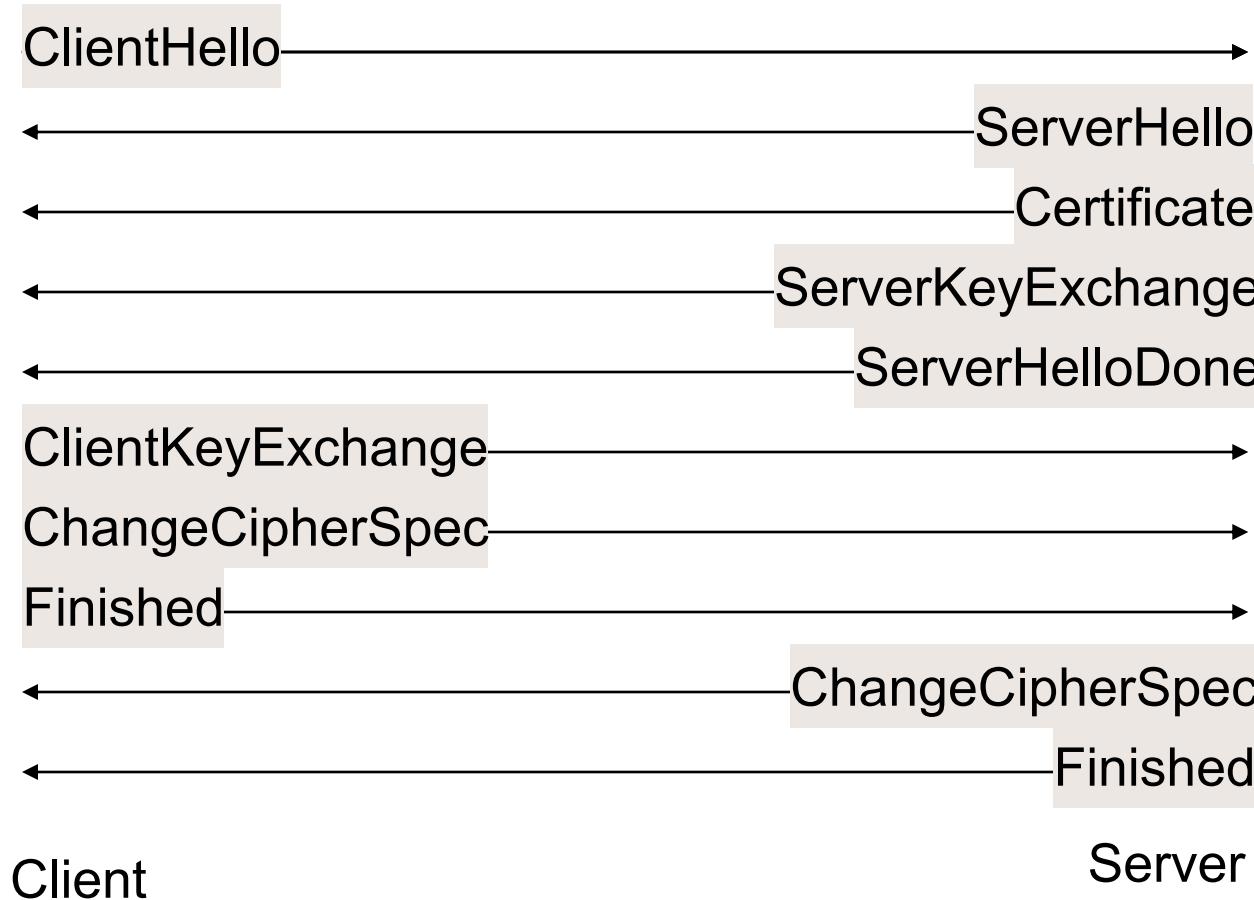
TLS Handshake Protocol (cont.)

- TLS Handshake Protocol message format:

1 byte	3 bytes	≥ 0 bytes
Type	Length	Content

- Currently defined TLS Handshake Protocol message types:
 - **client_hello (1)**
 - **server_hello (2)**
 - **certificate (11)**
 - **server_key_exchange (12)**
 - **certificate_request (13)**
 - **server_hello_done (14)**
 - **certificate_verify (15)**
 - **client_key_exchange (16)**
 - **finished (20)**

TLS 1.2 Handshake Protocol





TLS Handshake Protocol (cont.)

● ClientHello

- **the client starts the communication by sending the ClientHello message**
- **contains**
 - version number
 - random
 - optional session ID
 - used to resume a previous session
 - list of cipher suites supported
 - The cipher suite includes key exchange algorithm, symmetric algorithm (including chaining mode) and MAC algorithm



TLS Handshake Protocol (cont.)

- ServerHello
 - sent in response to the ClientHello message
 - with this message, the server finally decides which cipher suite to use
 - contains
 - version number
 - random
 - optional session ID (for resuming a previous session)
 - the cipher suite to be used, picked from the list of proposals given by the client
- Certificate
 - contains the server certificate, including the chain leading up to the CA root certificate
 - Optional according to the TLS specifications, but most (all?) implementations require a server certificate
 - If no certificate is sent, the ServerKeyExchange is required



TLS Handshake Protocol (cont.)

- ServerKeyExchange

- **used for the key exchange**

- Includes the server part of the key exchange
 - Exact meaning depends on the cipher suite chosen
 - For Diffie-Hellman, the modulus p , the generator g and $y = g^x$ is sent
 - Only when the server Certificate message (if sent) does not allow the client to generate a premaster secret
 - This is true for the following key exchange methods:
 - » DHE_DSS
 - » DHE_RSA
 - » DH_anon (removed)

- ServerHelloDone

- **marks the end of the server's part in the handshake**
 - It does not contain any other information



TLS Handshake Protocol (cont.)

- ClientKeyExchange
 - **always sent by the client**
 - **contains the client part in the key agreement**
 - the premaster secret is set, either
 - by direct transmission of the RSA-encrypted secret or
 - By the transmission of DH parameters
 - **the exact format depends on the exchange algorithm agreed on previously**
 - for Diffie-Hellman, the message contains the client's part
$$y = g^x \bmod p$$
- ChangeCipherSpec
 - **indicates that from this point, communication is encrypted**
- Finished
 - **is encrypted**
 - **marks the end of the handshake**



TLS Handshake Protocol (cont.)

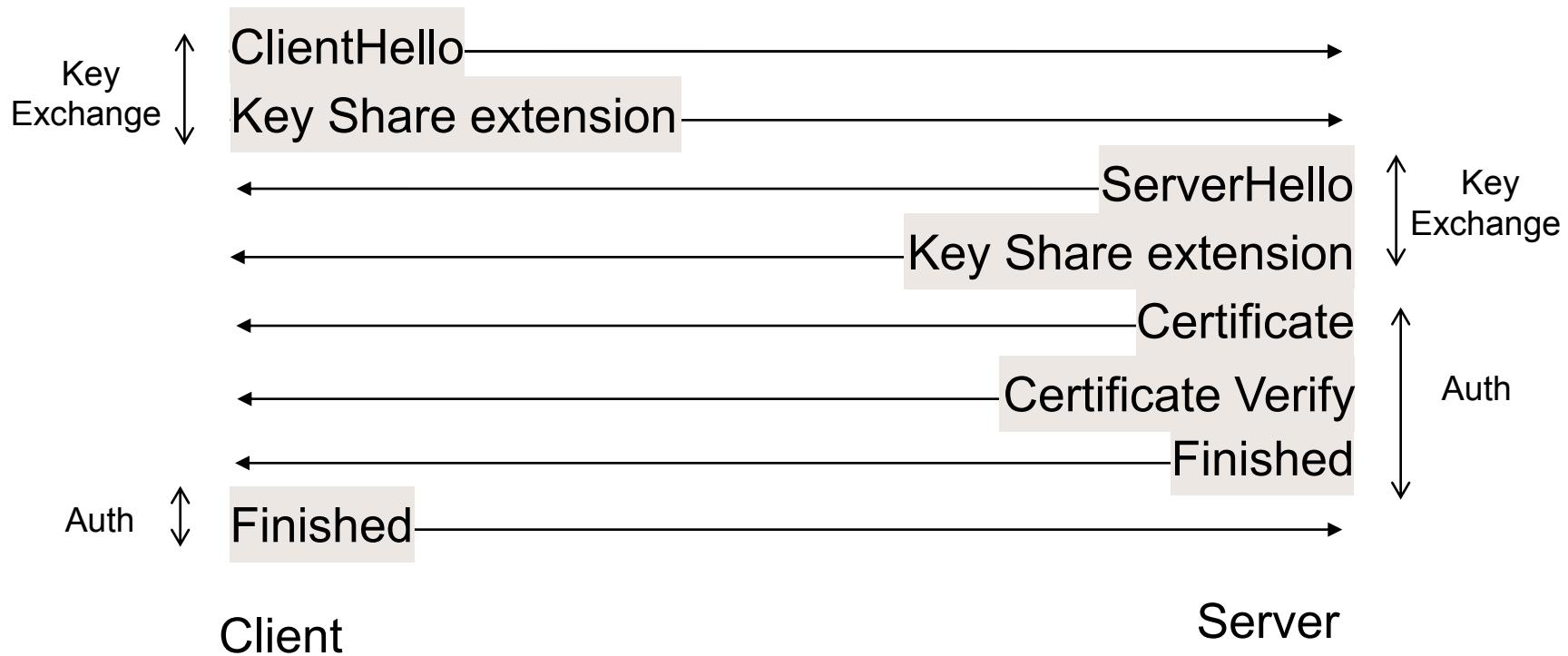
- ChangeCipherSpec and Finished
 - **play the same role as the client's message**
- After the handshake is complete, the client and the server start exchanging encrypted messages



Computing the Master Secret

- RSA
 - **48-byte pre_master_secret is generated by the client, encrypted under the server's public key, and sent to the server**
- DH or DHE
 - **A conventional Diffie-Hellman computation is performed**
 - The negotiated DH key is used as the PMSK

TLS 1.3 Handshake Protocol





TLS 1.3 Handshake Protocol

- In the Key Exchange phase
 - **client sends the ClientHello message**
 - random nonce, offered protocol versions, a list of symmetric ciphers
 - either a set of Diffie-Hellman key shares (in the "key_share" extension), a set of pre-shared key labels (in the "pre_shared_key" extension), or both
 - **server processes the ClientHello, determines the appropriate cryptographic parameters, and responds with ServerHello**
- The combination of the ClientHello and the ServerHello determines the shared keys
 - **if (EC)DHE key establishment is used, the ClientHello and ServerHello contain the ephemeral Diffie-Hellman values**
- In the Authentication phase
 - **in case of certificate-based authentication, the server sends its certificate and signature over the entire handshake using the private key**
 - **Finished messages contain a MAC over the entire handshake**
 - provide key confirmation, and bind the endpoint's identity to the exchanged keys



Generating secretes

- Key exchange methods establish a “Pre Master Secret” (PMSK) between the client and server
- For all key exchange methods, the same algorithm is used to convert the PMSK into the “Master Secret” (MSK) [48B]
 - **the PMSK should be deleted from memory once the MSK has been computed**

$\text{MSK} = \text{PRF}(\text{PMSK}, \text{"master secret"}, \text{ClientHello.random} \parallel \text{ServerHello.random})$

- **PRF:**
 - $\text{PRF}(\text{secret}, \text{label}, \text{seed}) = \text{P_hash}(\text{secret}, \text{label} \parallel \text{seed})$
 - $\text{P_hash}(\text{secret}, \text{data})$ is a data expansion function defined for expand a secret and seed into an arbitrary quantity of output:
 - $\text{P_hash}(\text{secret}, \text{data}) = H_1 \parallel H_2 \parallel H_3 \parallel \dots$
 - » $H_i = \text{HMAC_hash}(\text{secret}, A_i \parallel \text{data})$
 - » $A_0 = \text{data}$
 - » $A_i = \text{HMAC_hash}(\text{secret}, A_{i-1})$
 - P_hash can be iterated as many times as necessary to produce the required quantity of data



Generating secretes (cont.)

- MSK is used to generates secrete material such as
 - **a client write MAC secret,**
 - **a server write MAC secret,**
 - **a client write key,**
 - **a server write key,**
 - **a client write IV, and**
 - **a server write IV**
- MSK is hashed into a sequence of secure bytes

$\text{key_block} = \text{PRF}(\text{MSK}, \text{"key expansion"}, \text{server_random} \parallel \text{client_random})$

- assigned to the above secret material
 - **MAC secrets, encryption keys, IVs...**

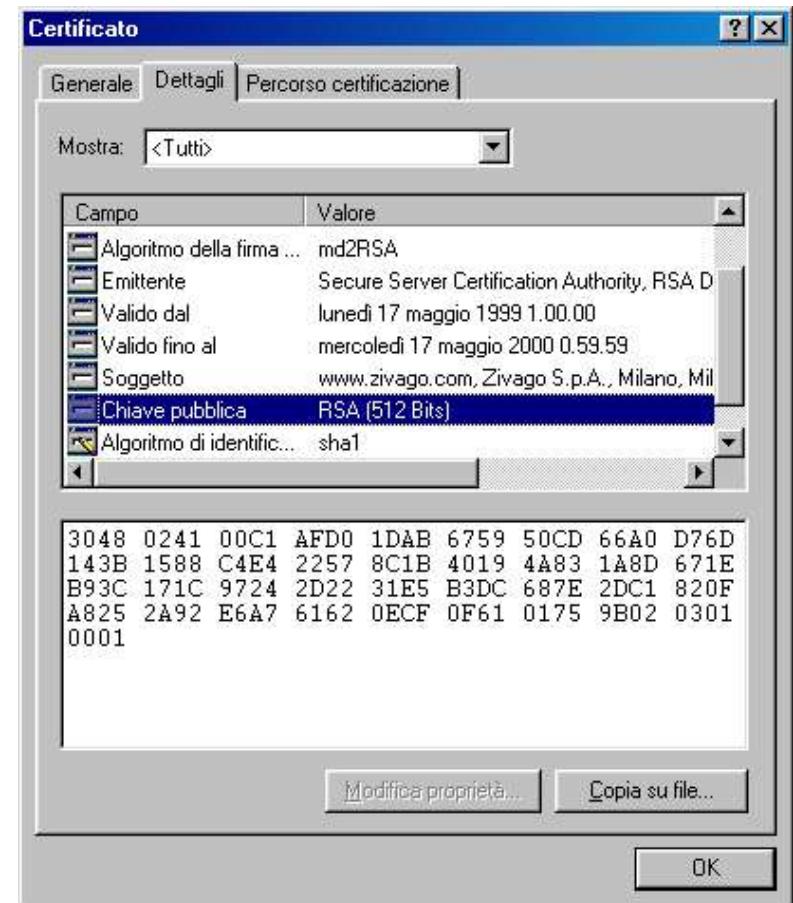


Verifying the certificate

- All certificates in TLS are in the X.509 format
- To verify that a certificate is valid, the verifier must
 - **Check that the CA signature is valid**
 - **Check that the owner of the certificate knows the private key**
 - **Check that the identifying information is what it should be**
- The protocol specifies how to perform the first two parts, but the last part is up to the implementation
- The CN field of the server certificate contains the host name of the server

Web Browser Support

- Some Root/CA certificates pre-installed
 - Firefox Opera and IE have different lists
 - CA's public key is used to verify the signatures on issued certificates
 - Browsers can accept unverifiable certificates or alert the user
- Users can install additional certificates
 - Additional Root/CA certificates
 - Security vulnerability
 - Client certificates





Datagram Transport Layer Security (DTLS)

- TLS cannot be directly used with UDP, since packets may be lost or reordered (unreliability)
 - **TLS does not allow independent decryption of individual records**
 - because the integrity check depends on the sequence number, if record N is not received, then the integrity check on record N+1 will be based on the wrong sequence number and thus will fail
 - prior to TLS 1.1, there was no explicit IV and so decryption would also fail
 - **TLS handshake layer assumes that handshake messages are delivered reliably**
 - breaks if those messages are lost
- Datagram Transport Layer Security (DTLS) Version 1.2
 - **TLS over datagram transport (UDP)**
 - **DTLS makes only the minimal changes to TLS required to fix the problem**



DTLS (cont.)

● Loss-Insensitive Messaging

- **In TLS Record Layer records are not independent**
 - Cryptographic context of stream cipher key stream is retained between records
 - Anti-replay and message reordering protection are provided by a MAC that includes a sequence number, but the sequence numbers are implicit in the records
- **DTLS solves the first problem by banning stream ciphers**
- **DTLS solves the second problem by adding explicit sequence numbers**



DTLS (cont.)

- Providing Reliability for Handshake
 - **the TLS handshake is a lockstep cryptographic handshake**
 - messages must be transmitted and received in a defined order; any other order is an error
 - this is incompatible with reordering and message loss
 - **DTLS uses a simple retransmission timer to handle packet loss**
 - **in DTLS, each handshake message is assigned a specific sequence number within that handshake**
 - when a peer receives a handshake message, it can quickly determine whether that message is the next message it expects
- TLS handshake messages are potentially larger than any given datagram, thus creating the problem of IP fragmentation
 - **In order to compensate for this limitation, each DTLS handshake message may be fragmented over several DTLS records, each of which is intended to fit in a single IP datagram**
 - each DTLS handshake message contains both a fragment offset and a fragment length



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Anonymity

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://www.tlc.unipr.it/veltri>



Introduction

- Each time we communicate through a network (e.g. the Internet), we send packets that contain information regarding where the message is going and who sent it (e.g., IP addresses)
 - anybody observing a link along the path can roughly identify who is communicating with, based on information contained in each packet
 - also in case of cryptography is used to protect the integrity and confidentiality of the contents of communication, the source and destination addresses may be still visible to an observer along the route on which a packet is traveling
 - this information often is enough to uniquely identify persons participating in a communication
 - sometimes these relationships as well as patterns of communication can be as revealing as their content
- The security service aiming to hide relationships between the communicating parties is called anonymity
- In general, privacy of a communication could require both data confidentiality and anonymity



Anonymity

- Can be defined as "the state of being not identifiable within a set of subjects, the anonymity set"
 - **the anonymity set is the set of all possible actors in a system that could have been the sender or recipient of a particular message**
- We could further refine the anonymity set based on the particular role subjects have with respect to a specific message
 - **sender anonymity set**
 - **recipient anonymity set**
- In general, anonymity systems seek to provide "unlinkability"
 - **between sent messages and their true recipients (recipient anonymity), and/or**
 - **between received messages and their true senders (sender anonymity)**



Adversaries – passive/active

- Adversaries of an anonymity system may be:

- **passive**

- is able to monitor and record the traffic on network links entering and exiting clients or servers in an anonymity network
 - this can enable powerful statistical attacks

- **active**

- has all the monitoring capabilities of a passive adversary
 - is also able to manipulate network traffic by controlling one or more network links or nodes
 - he or she can modify or drop traffic in parts of the network, as well as insert his own traffic or replay legitimate network traffic that he previously recorded



Adversaries – partial/global visibility

- The visibility of an adversary determines how much of the network he or she is able to passively monitor or actively manipulate
- An adversary of an anonymity system may be:
 - **a partial adversary**
 - is only able to monitor the links entering and exiting a particular node in the network, or a small, related subset of nodes in the network
 - an example of a partial adversary might be someone on the same LAN as a node in the anonymity network, or even a common ISP among multiple network nodes
 - **a global adversary**
 - is a powerful observer that has access to all network links in an anonymity network



Adversaries – internal/external

- An adversary may also be

- **external**

- does not participate in the anonymity network or its protocol
 - he or she can compromise the communication medium used by clients (i.e., their network links) in order to monitor or manipulate their network traffic

- **internal**

- is an active adversary that participates in the anonymity network protocol as a client, or operates a piece of the infrastructure, e.g. by running a node of the anonymity network



Anonymity systems

- Tools that attempt to provide anonymous communications, sometimes referred to also as "anonymizers">
 - **Protocol specific anonymizers**
 - implemented to work only with one particular application protocol
 - the advantage is that normally no application extension is needed
 - commands to the anonymizer are included inside a typical message of the given protocol
 - e.g. remailers and web proxies
 - **Protocol independent anonymizers**
 - protocol independence can be achieved by creating a tunnel to an anonymizer
 - protocols used by anonymizer services may include SOCKS, PPTP, or OpenVPN
 - in this case either the desired application must support the tunneling protocol, or a piece of software must be installed to force all connections through the tunnel
 - e.g. web browsers and FTP clients often support SOCKS



High latency vs Low latency

- Anonymity systems can often be classified into two general categories
 - **high-latency systems**
 - high-latency anonymity systems are able to provide strong anonymity, but are typically only applicable for non-interactive applications that can tolerate delays of several hours or more, such as email
 - sometimes referred to as message-based systems
 - **low-latency systems**
 - in contrast, low-latency anonymity systems usually provide better communication performance and are intended for real-time applications, particularly Web browsing
 - sometimes referred to as connection-based systems

High-latency anonymity systems: Mix

- A Mix is the basic building block of nearly all high-latency anonymity systems
- At a high level, it is a process that
 - accepts encrypted messages as input
 - groups several messages together into a batch, and
 - decrypts and forwards some or all of the messages in the batch
- For example, if Alice wants to anonymously send a message M to Bob via a single mix X
 - Alice sends $E_{K_X}(ID_B, M)$ to X, where
 - K_X is the public key of X, or a secret key shared between A and X
 - ID_B is Bob's address
 - for preventing an adversary from identifying two identical messages encrypted under the same key, a random value can be also included
 - $E_{K_X}(ID_B, M, R)$
 - the mix collects messages into a batch until it has received “enough”, and then
 - forwards each to the destination address extracted from the decrypted input message





Mix - Flushing algorithm

- The algorithm a mix uses to determine which messages to forward to their next destination and when to do so is often referred to as a “flushing algorithm”
 - **sometimes equivalently called “batching strategies”**
- There have been numerous flushing algorithms discussed in the literature, as well as possible active attacks against them
 - **Threshold Mixes**
 - simply collects incoming encrypted messages until it has received n messages, then it forwards them to their next destination in a random order
 - **Timed Mixes**
 - collects messages for a fixed length of time Δt
 - **Threshold and Timed Mixes**
 - a combination of the simple threshold and timed flushing algorithms
 - **Pool Mixes**
 - select a random subset of the collected messages to flush and then retain the rest in the mix for the next round
 - **Stop-and-Go Mixes**
 - SG-mixes individually delay messages as they pass through the mix

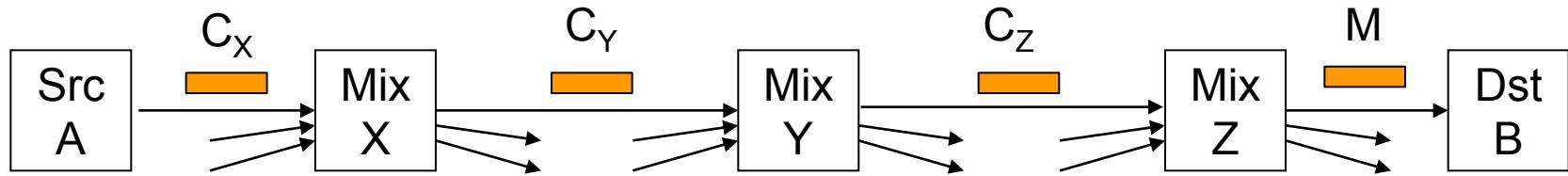


High-latency anonymity systems: Mix Networks

- Using a single mix leads not only to a single point of failure, but also to a single point of trust since a dishonest mix can reveal the true input-output correlations
- Instead of using a single mix, senders can choose an ordered sequence of mixes through which to send their messages
- If Alice wants to anonymously send a message M to Bob via a path P = {X, Y, Z}
 - **she would iteratively create a layer of encryption, in the same manner as above, for each mix starting with the last mix in the path and working back toward the first:**
 $E_{Kx}(ID_Y, E_{Ky}(ID_Z, E_{Kz}(ID_B, M)))$
 - **Alice then sends the resulting multiply encrypted ciphertext to the first mix in the path**
 - **each mix can remove a layer of encryption to extract the address of the next mix in the path and a ciphertext message to forward to that mix**
- To prevent the lengths of messages from leaking information about how many mixes a message has already passed through, mixes can pad the messages to a fixed length before forwarding them

Mix Networks (cont.)

- Example of Mix Network, with 3 Mixes X, Y, and Z:



- $C_X = E_{Kx}(ID_Y \parallel C_Y) = E_{Kx}(ID_Y \parallel E_{Ky}(ID_Z \parallel E_{Kz}(ID_B \parallel M)))$
- $C_Y = E_{Ky}(ID_Z \parallel C_Z)$
- $C_Z = E_{Kz}(ID_B \parallel M)$



Low-latency anonymity systems

- The flushing algorithms used by the mix-based systems above can introduce large delays on the order of several hours or even days between the time a message is sent and when it arrives at the intended recipient
 - **for applications like email, such delays are tolerable and often even expected**
 - **however, real-time, interactive applications, like Web browsing and SSH, require significantly lower latency**
- This lead to consider low-latency systems
 - **the improved performance of low-latency systems, make these systems more vulnerable to certain traffic analysis attacks**
 - **some types of low-latency systems:**
 - Anonymous Proxy
 - Anonymity Networks
 - Onion Routing



Anonymous Proxy

- Low-latency anonymity systems are often based on the notion of a “Proxy”
 - **in contrast to high-latency anonymity systems (based on mixes)**
 - **while mixes explicitly batch and reorder incoming messages, proxies simply forward all incoming traffic (e.g., a TCP connection) immediately without any packet reordering**
- It is a proxy server that acts as an intermediary between a client and the actual target server
 - **if the communication between the client and the proxy server is encrypted (e.g. via TLS) the client address is then only shared with the proxy server while the target server only sees the proxy server's address**
- Examples
 - **IP VPNs (e.g. OpenVPN)**
 - **HTTPS proxy servers**
- Proxies can also be used in chains forming a sort of Anonymity Networks



Onion Routing

- The most prevalent design for low-latency anonymous communications
- There is a set $R = \{R_1, R_2, \dots, R_n\}$ of servers called Onion Routers (ORs) that relay traffic for clients
- The first step is the establishment of an anonymous connection formed by multiply encrypted tunnel, or circuit, through the network
 - **sequence of k ORs**
 - **each OR stores encryption keys, connection IDs, prev/next OR IDs**
- The initiator first selects an ordered sequence of k ORs in the network to use as the circuit's path, much like in a mix network
 - **to minimize the computational costs, senders use public key cryptography to establish the circuit and then use faster symmetric key cryptography to transfer the actual data**
 - **the initiator generates two symmetric keys for each OR along the path**
 - a forward key K_F , used to encrypt data sent from the initiator towards the responder
 - a backward key K_B , applied to data from the responder to the initiator



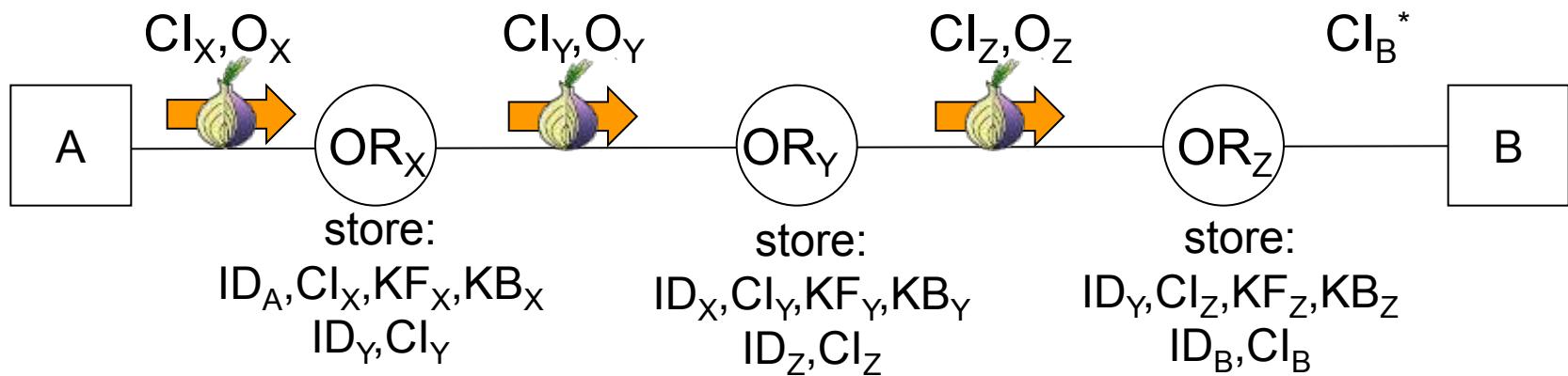
OR Circuit setup

- As an example, if the initiator chose the path $P = \{R_x, R_y, R_z\}$, it would construct the encrypted “onion”:
 - $E_{Kx}(t_x, KF_x, KB_x, R_y, E_{Ky}(t_y, KF_y, KB_y, R_z, E_{Kz}(t_z, KF_z, KB_z, ID_B^*, \emptyset)))$
 - values t_i indicate the expiration time of the onion
 - \emptyset means no inner onion is present, indicating to R_z that it is the last router in the path
 - optionally, the address of B (ID_B) can be included in the inner onion in order to establish a connection to node B
- The initiator sends the onion, together with a chosen connection ID CI_x , to the first OR in the path, R_x
 - which removes from the onion the outermost layer of encryption using its private key, and learns the symmetric keys KF_x and KB_x generated by the initiator, as well as the next server in the path
 - R_x then pads the remaining encrypted payload with random bytes, so the onion maintains a constant length, and sends the result, together with a new connection ID CI_y , to R_y

OR Circuit setup (cont.)

- Each OR along the path:
 - repeats this process, until the onion reaches the end of the path
 - knows its predecessor and successor but no other node in the circuit
 - store in a local DB the two connection IDs, the two neighbors' addresses, and the two keys

- Example:
 - $O_x = E_{Kx}(KF_x \parallel KB_x \parallel R_Y \parallel O_Y)$
 - $O_Y = E_{Ky}(KF_Y \parallel KB_Y \parallel R_z \parallel O_z)$
 - $O_z = E_{Kz}(KF_z \parallel KB_z \parallel B)$





Onion Routing (cont.)

- Once the circuit is constructed, the initiator can relay its application traffic over the circuit using the symmetric keys generated for each hop
- Similarly to how the onion was constructed, the initiator
 - encrypts a message m once for each router in the circuit using the generated forward keys KF_i , and then
 - sends it, together with the first CI_1 , to the first router in the circuit
 - data sent from A to the first OR R_1 is: $CI_1, E_{KF1}(E_{KF2}(E_{KF3}(\dots(E_{KF_n}(m))))$
- Each OR R_i in the circuit can remove a layer of encryption and forward the result, together with the corresponding output CI_{i+1} , to the next OR_{i+1} , until it reaches the last OR, which can then forward the original message m to the destination
- When a response is sent along the reverse path, a layer of encryption is added by each OR in the circuit using the backward key KB_i
- The initiator can then remove all layers of encryption to recover the original response, since he or she has the secret keys that she or he previously generated for each OR

Tor

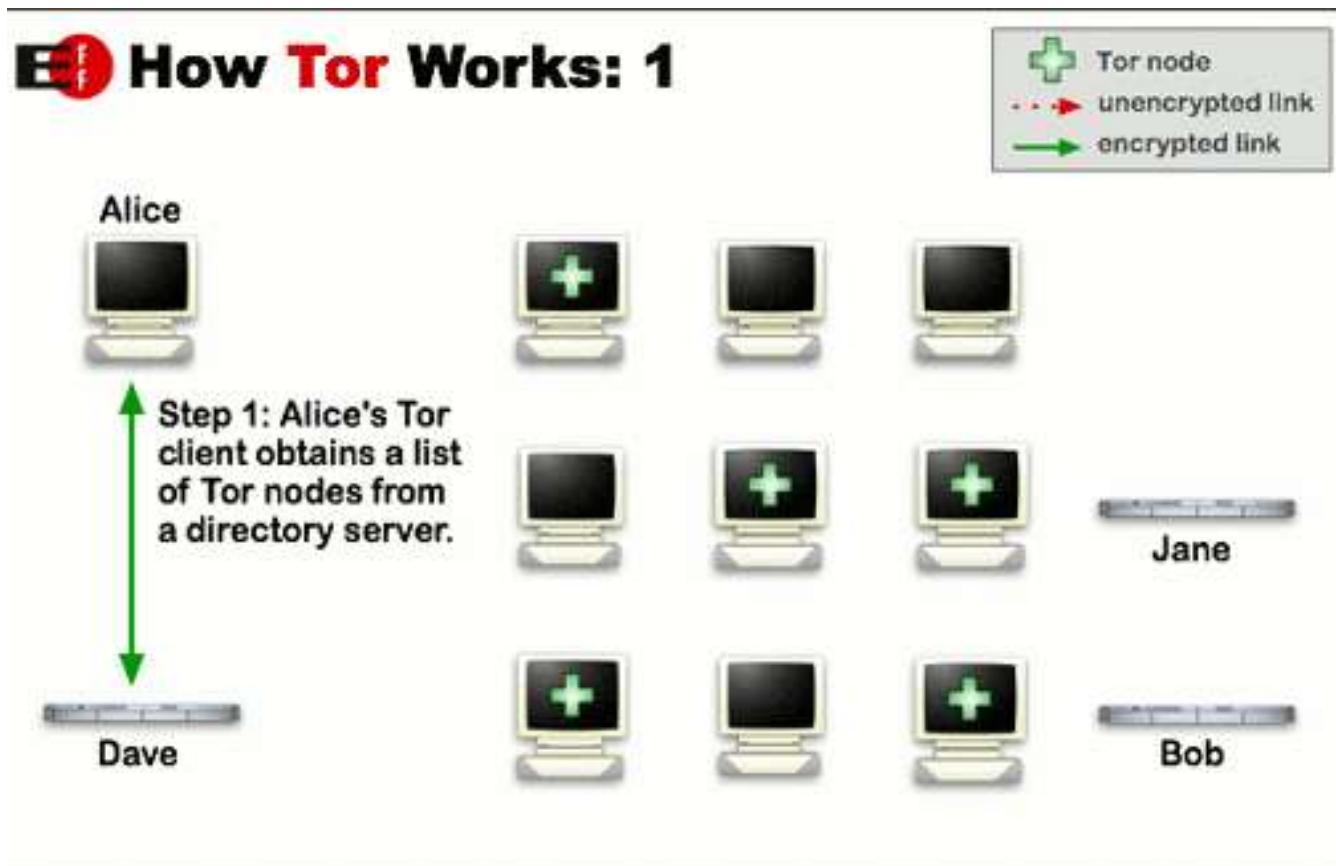


Tor

- Tor (The Onion Router) is a distributed overlay network designed to anonymize low-latency TCP-based applications such as web browsing, secure shell, and instant messaging
 - **originally developed by the U.S. Naval Research Laboratory**
- (Main) Onion routing system
 - **Tor clients running an onion proxy (OP) choose a path through the Tor network and build a “circuit”**
 - the OP periodically negotiates the virtual circuit using multi-layer encryption, ensuring perfect forward secrecy
 - each node (OR) in the path knows its predecessor and successor, but no other nodes in the circuit
 - **the OP presents a SOCKS interface to its clients**
 - SOCKS-aware applications may be pointed at Tor, which then multiplexes the traffic (TCP streams) through a Tor virtual circuit
 - **traffic flowing down the circuit is sent in fixed-size “cells”, which are unwrapped by a symmetric key at each node and relayed downstream**

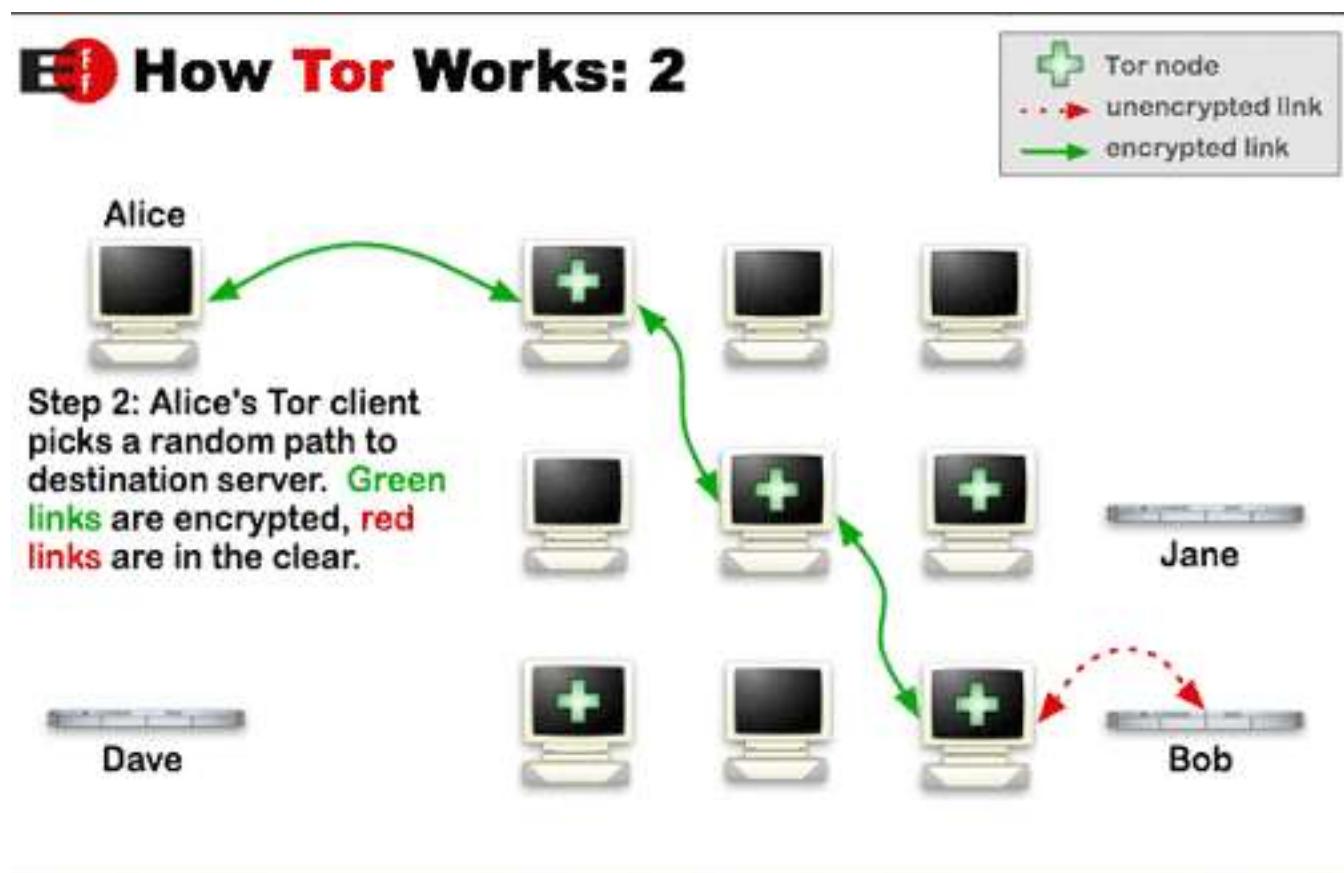


How Tor works

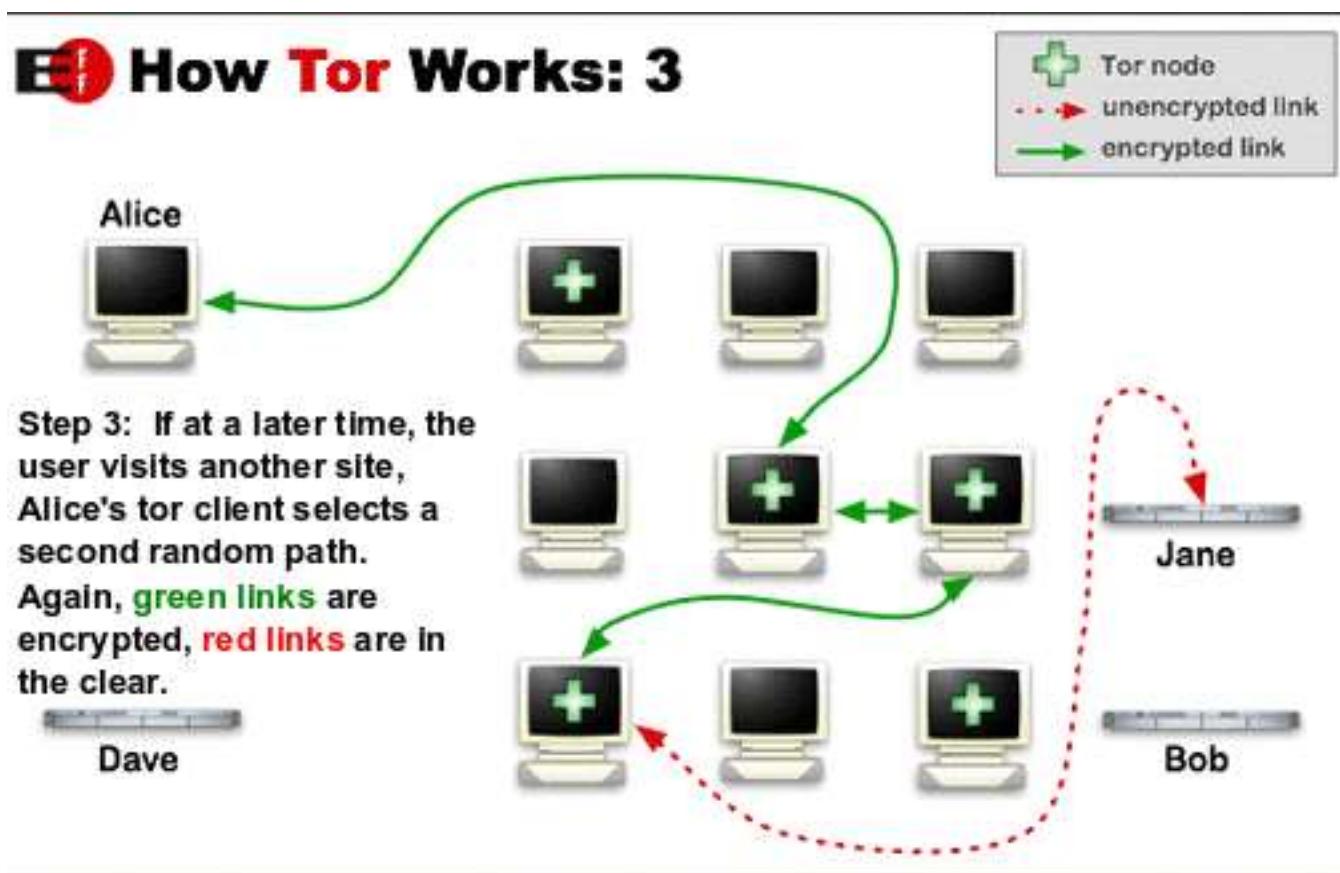




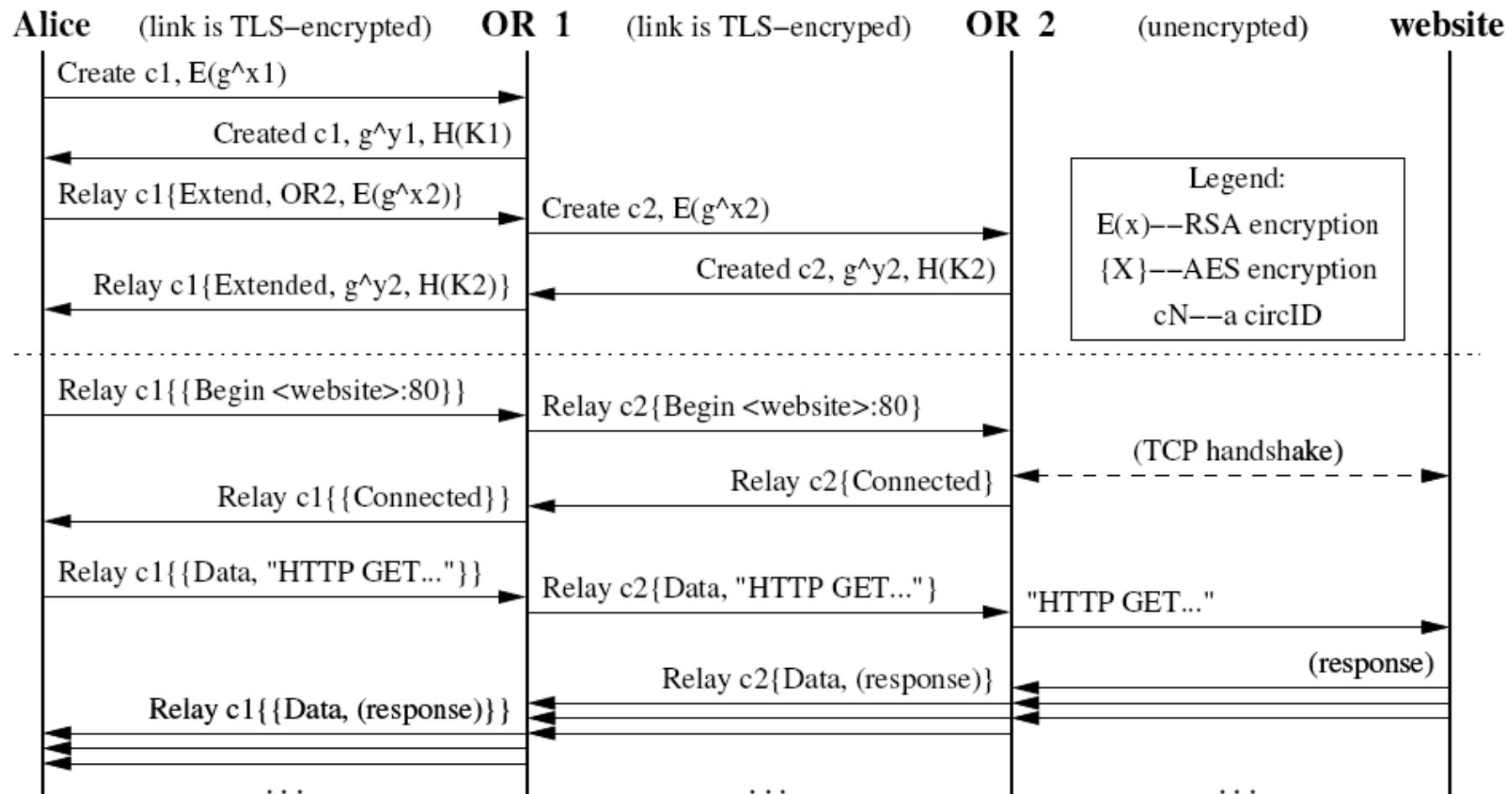
How Tor works (cont.)



How Tor works (cont.)



Tor circuits and streams





Tor circuits and streams (cont.)

- A user's OP constructs circuits incrementally, negotiating a symmetric key with each OR on the circuit, one hop at a time
 - **to begin creating a new circuit, the OP (call her Alice) sends a `create` cell to the first node in her chosen path (call him OR1)**
 - Alice chooses a cirID C_1 for the new circuit
 - **the `create` cell's payload contains the first half of the Diffie-Hellman handshake g^{x_1} , encrypted to the onion key of OR1**
 - **OR1 responds with a `created` cell containing g^{y_1} along with a hash of the negotiated key $K_1 = g^{x_1 y_1}$**
- To extend the circuit further:
 - **Alice sends a `relay extend` cell to OR1, specifying the address of the next OR (OR2), and an encrypted g^{x_2} for OR2**
 - **OR1 copies the half-handshake into a `create` cell, and passes it to OR2 to extend the circuit**
 - OR1 associates C_1 to a new chosen cirID C_2
 - **when OR2 responds with a `created` cell, OR1 wraps the payload into a `relay extended` cell and passes it back to Alice**
 - **now the circuit is extended to OR2, and Alice and OR2 share a common key $K_2 = g^{x_2 y_2}$**



Tor circuits and streams (cont.)

- To extend the circuit to other node or beyond, Alice proceeds as above, always telling the last node to extend one hop further
- Circuit-level handshake and key agreement:
 - Alice → OR_i : E_{K⁺}_{OR1}(g^{x_i})
 - OR_i → Alice : g^{y_i}, H(K_i || "handshake")
- This protocol achieves:
 - **unilateral entity authentication**
 - in the second step, OR_i proves that it was he who received g^{x_i} , and who chose y_i
 - Alice knows she's handshaking with the OR_i, but the OR doesn't care who is opening the circuit
 - » Alice uses no public key and remains anonymous
 - **unilateral key authentication**
 - Alice and OR_i agree on a key, and Alice knows only the OR learns it
 - **forward secrecy and key freshness**



Tor circuits and streams (cont.)

- Once Alice has established the circuit (so she shares keys with each OR on the circuit), she can send relay cells
- Upon receiving a relay cell, an OR looks up the corresponding circuit, and decrypts the relay header and payload with the session key for that circuit



Tor circuits and streams (cont.)

- When Alice's application wants a TCP connection to a given address and port, it asks the OP (via SOCKS) to make the connection
 - **the OP chooses the newest open circuit (or creates one if needed), and chooses a suitable OR on that circuit to be the exit node (usually the last node)**
 - **the OP then opens the stream by sending a RELAY Begin cell to the exit node, using a new random streamID**
 - **once the exit node connects to the remote host, it responds with a RELAY Connected cell**
 - **upon receipt, the OP sends a SOCKS reply to notify the application of its success**
- The OP now accepts data from the application's TCP stream, packaging it into relay data cells and sending those cells along the circuit to the chosen OR



Tor: Hidden services

- Tor can also provide anonymity to websites and other servers
 - **these servers are configured to receive inbound connections through Tor**
 - **they are called hidden services**
- Because hidden services do not use exit nodes, they are not subject to exit node eavesdropping



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Vulnerabilities and attacks

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://www.tlc.unipr.it/veltri>



Vulnerabilities

- Vulnerability (Def.):
 - [NIST Glossary]: "**Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source**"
 - [IETF RFC4949]: "**A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy**"
- The exploitation of a system vulnerability may let an attacker to carry out unauthorized actions against the Confidentiality, the Integrity or the Availability of the system related assets



Vulnerabilities (cont.)

- Most systems have vulnerabilities, however not every vulnerability results in an attack, and not every attack succeeds
 - **success depends on the degree of vulnerability, the strength of attacks, and the effectiveness of any countermeasures in use**
 - **if the attacks needed to exploit a vulnerability are very difficult to carry out, then the vulnerability may be tolerable**
- Useful archives:
 - **Common Weakness Enumeration (CWE)**
 - <https://cwe.mitre.org>
 - list of common software and hardware security weakness types
 - **Common Vulnerabilities and Exposures (CVE)**
 - <https://cve.org>
 - search: https://cve.mitre.org/cve/search_cve_list.html
 - list of publicly known cybersecurity vulnerabilities
 - each entry contains an identification number and a description



Vulnerabilities (cont.)

- Vulnerabilities can be distinguished based on:

- **nature**

- unintentional (e.g. bugs and flaws)
 - intentional (e.g. backdoors)

- **domain**

- technology
 - design or specification
 - software/hardware implementation
 - operation and management
 - Inadequacy of organizational aspects in terms of: defense infrastructures, attacks detections, incident response
 - Ineffective security strategy in terms of: best practices, tools, technologies

- human

- bad behaviors
 - social engineering
 - » psychological manipulation of people into performing actions or divulging confidential information



Technology vulnerabilities

- Network and communication protocol vulnerabilities
 - **protocol specification flaws**
 - **protocol implementation flaws**
 - **misuses**
 - e.g. unsecure communication protocols, exploitation of address resolution mechanisms (e.g. ARP or DNS), dynamic configuration protocols (BOOTP/DHCP), etc.
- Software and hardware vulnerabilities
 - **Application implementation flaws**
 - **Operating system flaws**
 - OSs are the main sources of reported system vulnerabilities
 - **Hardware flaws**



From vulnerabilities to attacks

- If a vulnerability in a system is known or discovered and reachable, it can lead to an attack
 - **exploitation of the vulnerability**
- The success of an attack can further lead to another attack
- Examples of vulnerabilities that lead to different types of attacks:
 - **Open ports on outward facing Web and other servers, and code listening on those ports**
 - **Services available on the inside of a firewall**
 - **Code that processes incoming data, email, XML, office documents, and industry-specific custom data exchange formats**
 - **Interfaces, SQL, and Web forms**
 - **An employee with access to sensitive information vulnerable to a social engineering attack**



Attack Surfaces

- Network Attack Surface
 - **Vulnerabilities over an enterprise network, wide-area network, or the Internet**
 - **Included in this category are network protocol vulnerabilities, such as those used for a denial-of-service attack, disruption of communications links, and various forms of intruder attacks**
- Software Attack Surface
 - **Vulnerabilities in application, utility, or operating system code**
 - **Particular focus is Web server software**
- Human Attack Surface
 - **Vulnerabilities created by personnel or outsiders, such as social engineering, human error, and trusted insiders**

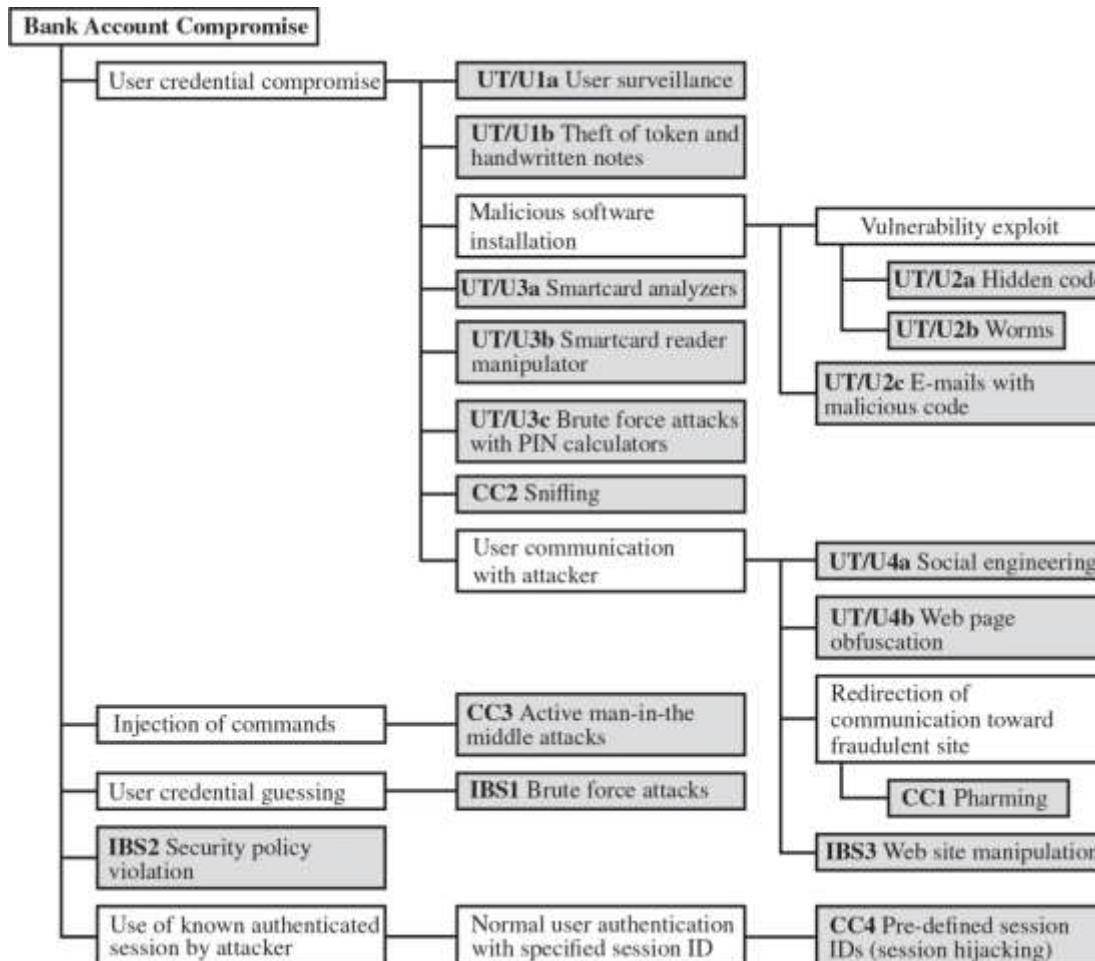


Attack Tree

- A branching hierarchical data structure that represents a set of potential techniques for exploiting security vulnerabilities
 - **the security incident that is the goal of the attack is represented as the root node of the tree**
 - **the ways that an attacker could reach that goal are iteratively and incrementally represented as branches and subnodes of the tree**
- Each subnode defines a subgoal, and each subgoal may have its own set of further subgoals, etc
- The final nodes on the paths outward from the root, represent different ways to initiate an attack
- Each node other than a leaf is either an AND-node or an OR-node
 - **branches can be labeled with values representing difficulty, cost, or other attack attributes, so that alternative attacks can be compared**

Example of an Attack Tree

- Example of an Attack Tree for internet banking authentication:



- UT/U: attack targets the user terminal, including smartcards, and actions of the user
- CC: attack focuses on communications channel
- IBS: attacks against the Internet banking servers



Computer Security Strategy

- A comprehensive security strategy involves three aspects:
 - **Security Policy**
 - formal statement of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources
 - **Security Implementation**
 - involves four complementary courses of action:
 - Prevention
 - Detection
 - Response
 - Recovery
 - **Assurance and evaluation**
 - encompasses both system design and system implementation
 - assurance is an attribute of an information system that provides grounds for having confidence that the system operates such that the system's security policy is enforced
 - evaluation is the process of examining a computer product or system with respect to certain criteria
 - involves testing and may also involve formal analytic or mathematical techniques



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Network vulnerabilities

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://www.tlc.unipr.it/veltri>



Network Attacks

- There are different typologies of attacks that can occur in a networked scenario
- Generally, objectives of an attack include:
 - **impersonification of an entity**
 - violation of peer entity authenticity and/or data
 - **fraudulent gathering of personal data**
 - violation of confidentiality (and sometimes peer entity authenticity)
 - **damaging or denying of a service**
 - violation of availability



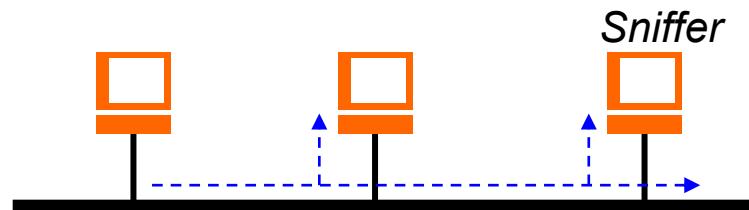
Network Attacks (cont.)

- Attack techniques include:

- **data interception**
 - network eavesdropping (sniffing)
 - spoofing
 - redirection and routing attacks
- **violation of authentication (and encryption) systems**
 - password cracking, guessing
- **denial of service (against a network service or resource)**
 - e.g. via distributed attack
 - network flooding, distributed denial of service (DDoS)
- **network and vulnerabilities scanning**
- **exploitation of vulnerabilities**
 - protocol vulnerabilities
 - software vulnerabilities
- **running malware**

Network eavesdropping

- Network eavesdropping or network sniffing is an attack consisting of capturing packets from the network transmitted by others' nodes and reading the data content
 - **in search of sensitive information like passwords, session tokens, or any kind of confidential information**
- The attack could be done by using tools called network sniffer (or protocol analyzers)
 - **these tools collect packets on the network and, depending on the quality of the tool, analyze the collected data like protocol decoders or stream reassembling**
- Can work in passive mode
 - **packets are simply captured, copied, and passed at user level for further analysis**
 - **requires to be along the path or in a broadcasting domain**





LAN eavesdropping

- In case of a LAN, the attacker configures the network interface for working in promiscuous mode capturing all packets passing through the network interface
 - **LAN with hub**
 - the hub duplicates every frame to all interfaces (ports)
 - **switched LAN**
 - a switch by default only transmits a frame to the destination port
 - there are attacks that may
 - make the device to switch from partitioning to broadcasting mode (fail-open mode), e.g. MAC flooding
 - redirect the traffic from one port to another; then the system may act like a router between the source and destination (Man-In-The-Middle attack), e.g. arp spoof attack
 - **wireless LAN**
 - sniffing is possible if no encryption is used or if a crack attack is performed against the WiFi key



Network sniffer

- Most common network sniffers (or protocol analyzer) are:
 - **WireShark**
 - GUI
 - **tcpdump (unix)**
 - command-line
- Windows sniffers usually are based on winpcap or npcap DLL libraries
- Unix sniffers usually are based on the tcpdump libpcap library
- Usually they require root/admin privileges



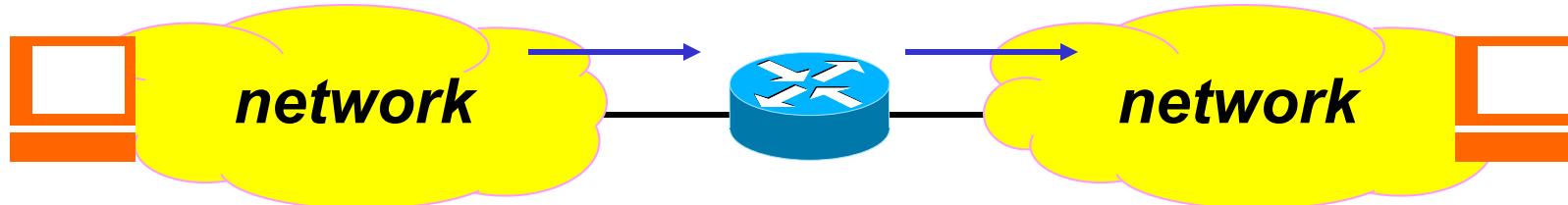
Network sniffer detection

- The presence of a network sniffer may be detected by means of
 - **proper commands directly on the node used to run it, e.g.:**
 - ifconfig

```
eth0 Link encap:Ethernet HWaddr 00:10:4B:E2:F6:4C
        inet addr:192.168.1.8 Bcast:192.168.1.255 Mask:255.255.255.0
              UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
              RX packets:1016 errors:0 dropped:0 overruns:0 frame:0
              TX packets:209 errors:0 dropped:0 overruns:0 carrier:0
              collisions:0 txqueuelen:100
```
 - cpm (Check Promiscuous Mode)
 - ifstatus
 - **specific sniffing detection tools that exploit unusual behaviours of the OS when working in promiscous mode**

Man-In-The-Middle attack (MITM)

- Using a network sniffer in a LAN is not the only mechanism for packet interception
- Packet interception can be also performed by means of
 - intercepting packet in a node already along the path
 - exploiting vulnerabilities of routing and/or address resolution protocols
 - e.g. ARP spoofing, ICMP redirect, DNS record poisoning, etc.
- In these cases, the attack is called Man-In-The-Middle (MITM)
 - the attacker is also able to modify the packets





Spoofing attack

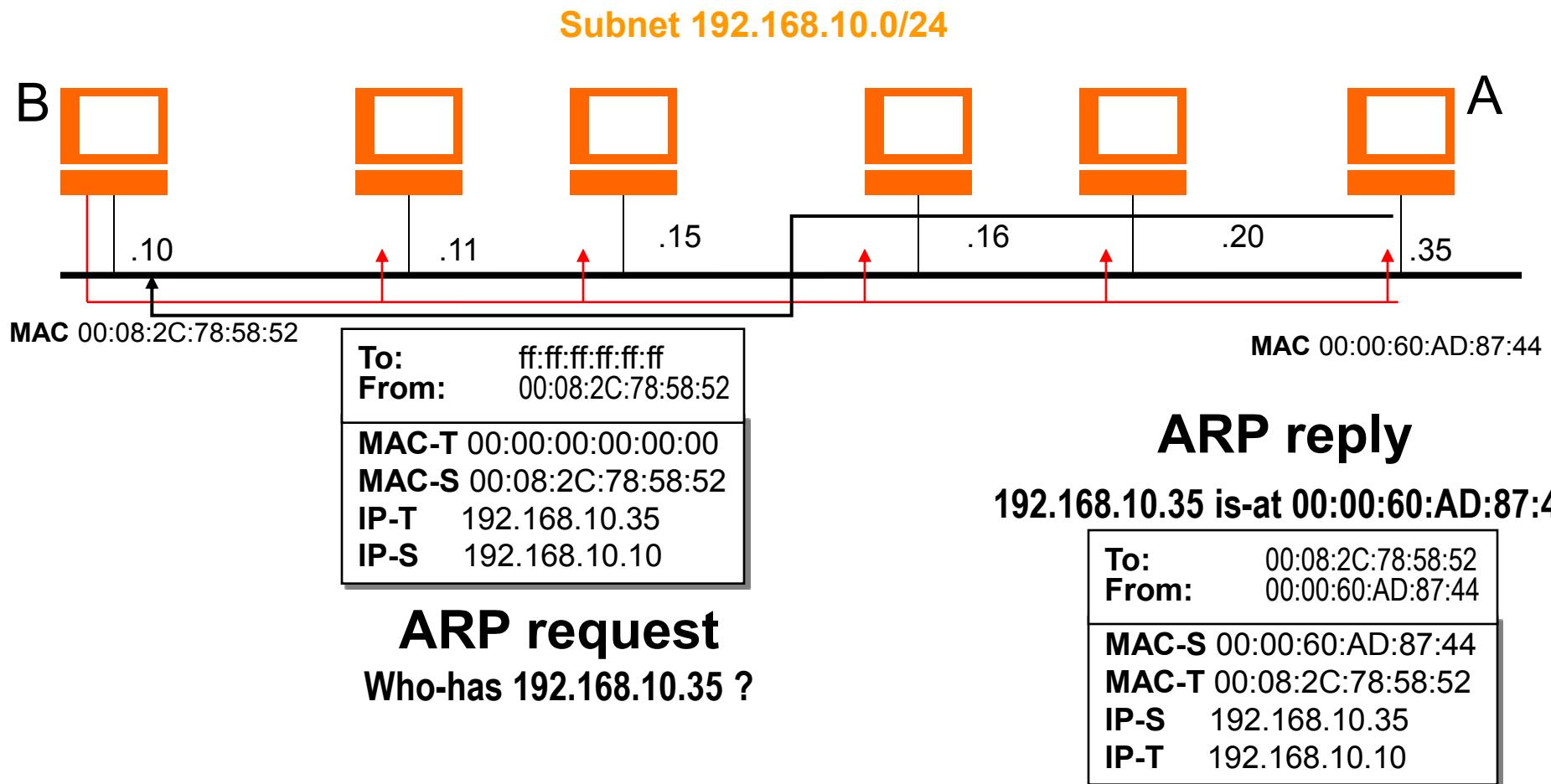
- A technique where one entity (attacker) successfully masquerades as another by falsifying data origin (spoofing)
- It could be performed at different layers
 - **Layer 2 (Link)**
 - MAC spoofing, ARP spoofing
 - **Layer 3 (IP)**
 - IP spoofing
 - **Layer 4 (Transport)**
 - UDP or TCP spoofing
 - **Layer 7 (Application layer)**
 - e-mail address spoofing
 - VoIP Caller ID spoofing
 - webpage spoofing (also known as phishing)
- When the fake identity (spoofed) is a real identity, it leads to social engineering attacks



ARP Spoofing

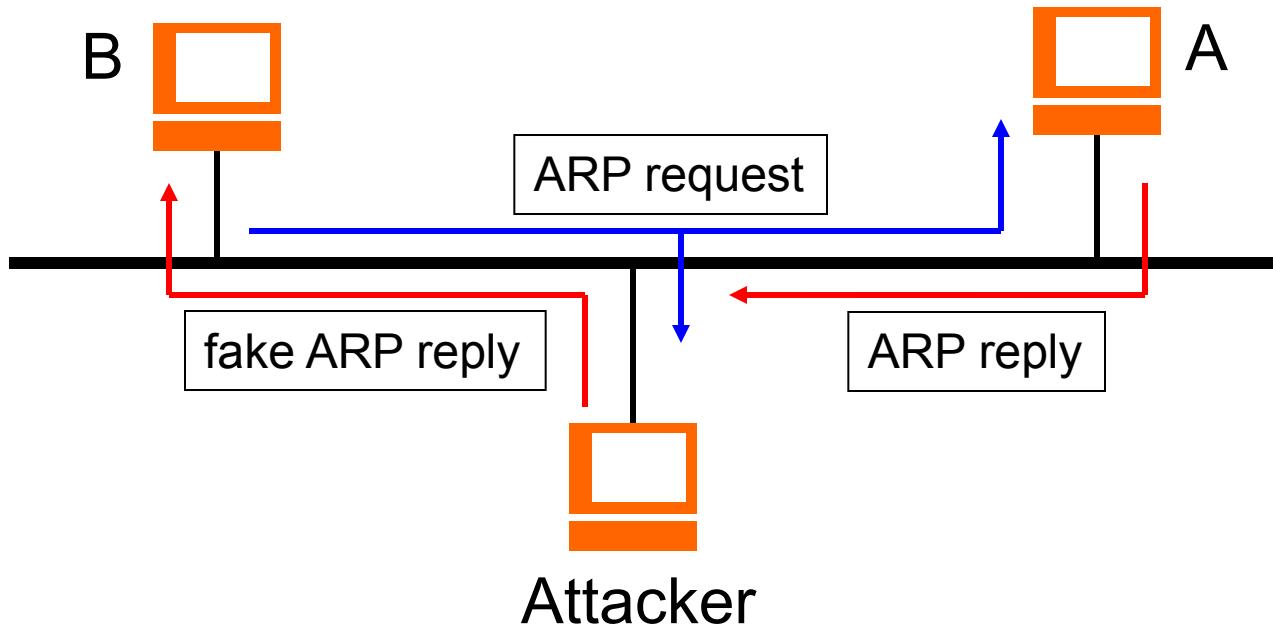
- An attacker sends fake ("spoofed") Address Resolution Protocol (ARP) messages onto a Local Area Network
 - generally, the aim is to associate the attacker's MAC address with the IP address of another host (e.g. the default gateway), causing any traffic meant for that IP address to be sent to the attacker instead
 - also referred to as ARP poisoning
 - can only be used on networks that make use of the ARP protocol and is limited to local network segments
- ARP spoofing allows an attacker to intercept data frames on a LAN, modify the traffic, or stop the traffic altogether
 - often used as an opening for other attacks, such as man in the middle, session hijacking, or denial of service attacks
- ARP spoofing may also be used as non-attacking technique
 - E.g. by Captive Portals, or in Mobile IP

Normal ARP protocol behaviour



- When receiving the replay, the client adds the pair {MAC_addr, IP_addr} to its ARP table

ARP Spoofing (cont.)





Defenses against ARP spoofing

- Static ARP entries
 - IP-to-MAC mappings in the local ARP cache can be statically defined, and then hosts can be directed to ignore all ARP reply packets
 - however it results in a big effort for maintenance
- ARP spoofing detection software
 - it detects ARP spoofing based on cross-checking of ARP responses
 - the existence of multiple IP addresses associated with a single MAC address may indicate an ARP spoof attack, although there are legitimate uses of such a configuration
 - may be implemented in individual hosts or may be integrated into Ethernet switches or other network equipment
- OS security
 - operating systems may react differently



MAC Spoofing - Port stealing

- Another type of MAC spoofing attack
- It floods the LAN with packets with a spoofed source MAC address
 - e.g. by sending many layer-2 packet to its real MAC address
 - this process "steals" the switch port of the spoofed host
 - packets destined to "stolen" MAC addresses will be received by the attacker, winning the race condition with the real port owner
 - when the attacker receives packets for "stolen" hosts, it may stop the flooding process and performs an ARP request for the real destination of the packet
 - when it receives the ARP reply it's sure that the victim has "taken back" his port
- This technique is useful to sniff in a switched environment when ARP poisoning is not effective (for example where static mapped ARPs are used)



IP Spoofing

- Creation of Internet Protocol (IP) packets with a forged source IP address
 - **most frequently used in MITM or denial-of-service attacks (DoS/DDoS)**
 - **also a method of attack used by network intruders to defeat network security measures, when trust relationships exist between machines based on IP addresses, e.g.:**
 - RPC services
 - X Window System
 - the R services suite (rlogin, rsh, etc.)
 - any service that uses IP address authentication
 - **sometimes used to by-pass firewalls (e.g. using the IP source routing option)**



TCP spoofing - TCP session hijacking

- Technique for establishing or intercepting a TCP session (connection) between two machines
- Two possibilities:
 - **source routing**
 - the initial hijacking method used involved using the source routing option of the IP protocol
 - this option made it possible to specify the path IP packets were to follow
 - The destination reverses the path in the opposite direction
 - **blind attack**
 - the only possible technique when source routing is disabled and the attacker is not in the same network of the two machines
 - involves sending packets as "blind attacks"
 - requires the establishment of a TCP connection (3-way handshake) without receiving responses, by trying to predict sequence numbers
 - more complex than an IP spoofing attack

TCP Spoofing (cont.)

- (Spoofed) TCP 3-way handshake

H >	SYN	> B	start of connection setup
A <	SYN/ACK	< B	response of the victim, set to the spoofed host
H >	ACK	> B	handshake completed
H >	PSH	> B	data

- TCP uses 32 bit Seq/Ack numbers

- these numbers make it relatively hard to spoof the source address because successful spoofing requires guessing the correct initial sequence number (ISN) which is generated by the server in a non-guessable way
- it is commonly known that a 32 bit number can be brute forced in a couple of hours given a fast (gigabit) network connection



Denial of Service (DoS)

- Attack for making a machine or network resource unavailable to its intended users
 - **most common technique is flooding**
 - It consumes victim resource (network bandwidth and/or processing capacity)
- Can be executed by
 - **a single node**
 - **two or more nodes**

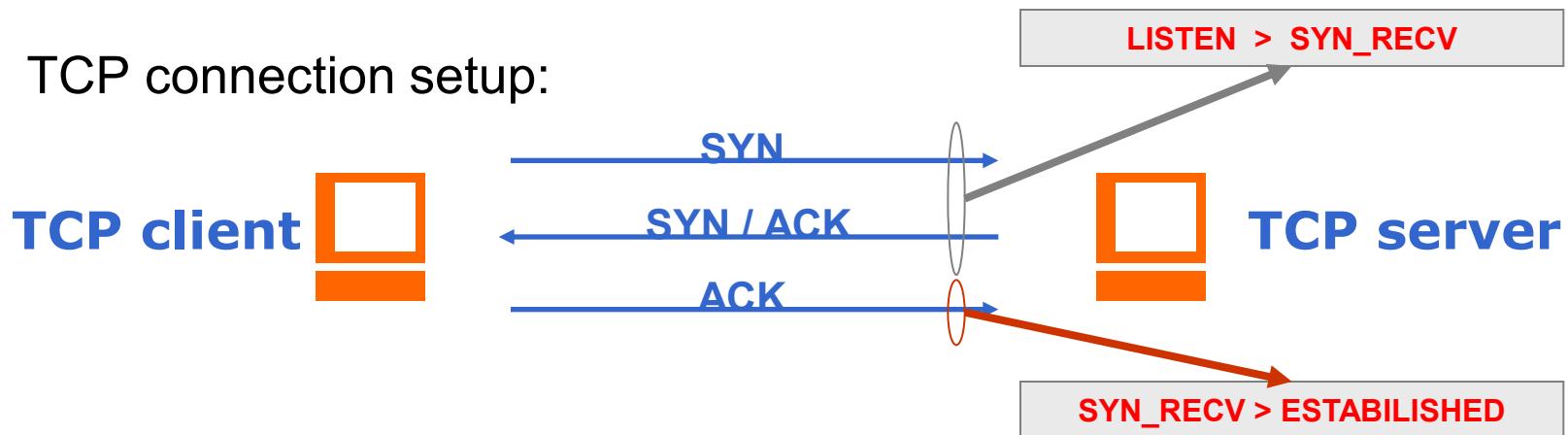


DoS (cont.)

- The attack is successful in consuming resources on the victim by either:
 - **using a computer with greater computation power and/or large network bandwidth**
 - **taking advantage of a property of the victim system which enables an attack consuming vastly more of the victim's resources than the attacker's (an asymmetric attack)**
 - e.g. SYN flood
 - **using a large number of computers and directing them to attack as a group**
 - Distributed Denial of Service (DDoS) attack
 - a set (or network) of nodes controlled by a single entity
 - e.g. (ICMP) Smurf, botnet
- An attack may utilize a combination of these methods in order to magnify its power

DoS: SYN flood

- TCP connection setup:

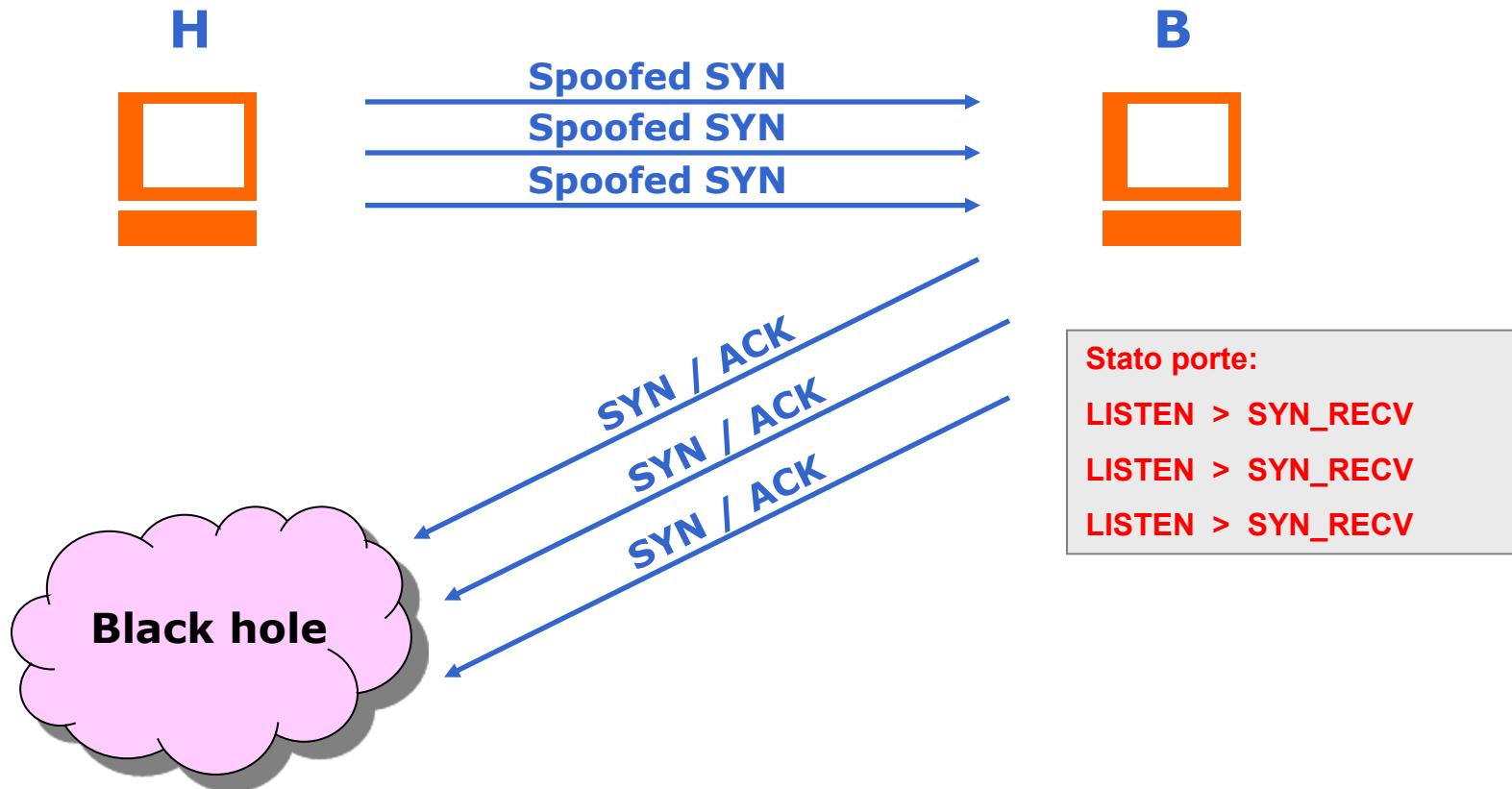


- TCP SYN flooding attack:

- A host sends a flood of TCP/SYN packets, often with a forged sender address
 - each of these packets is handled like a connection request, causing the server to spawn a half-open connection, by sending back a TCP/SYN-ACK packet (Acknowledge), and waiting for a packet in response from the sender address (response to the ACK Packet)
 - however, because the sender address is forged, the response never comes
- These half-open connections saturate the number of available connections the server is able to make, keeping it from responding to legitimate requests until after the attack ends

DoS: SYN flood (cont.)

- TCP SYN flooding attack:



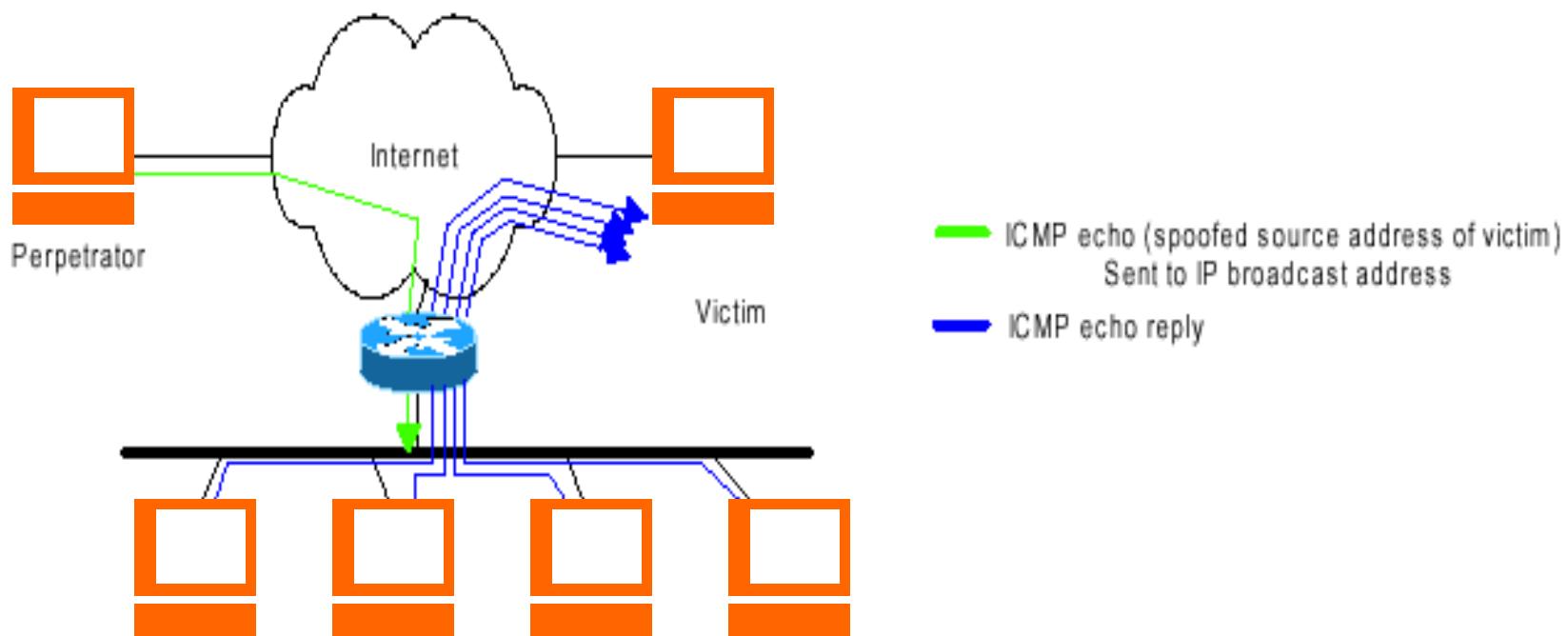


DDoS: Smurf

- A Smurf attack is a flooding DoS exploiting the possibility of sending packets to all computer hosts on a particular network via the broadcast address of the network
 - the attacker sends large numbers of packets with the source address faked to appear to be the address of the victim
 - the network then serves as a smurf amplifier
 - the network's bandwidth is quickly used up, preventing legitimate packets from getting through to their destination
 - it relies on misconfigured network devices that allow the forwarding of broadcast packets

DDoS: ICMP/Ping Flooding (Smurf)

- Ping flood is based on sending the victim an overwhelming number of ping packets





DoS Attack Defenses

- These attacks cannot be prevented entirely
- High traffic volumes may be legitimate
- Four lines of defense against DDoS attacks
 - **Attack prevention and preemption**
 - before attack
 - **Attack detection and filtering**
 - during the attack
 - **Attack source traceback and identification**
 - during and after the attack
 - **Attack reaction**
 - after the attack



Routing attacks

- Attacks that use vulnerabilities of dynamic configuration or routing mechanisms in order to redirect or block network traffic
- They may exploit:
 - **address resolution mechanisms (e.g. ARP or DNS)**
 - **Dynamic configuration protocols (BOOTP/DHCP)**
 - **routing protocols**
- They may operate at different protocol stack level:
 - **Layer 2 (Link)**
 - e.g. ARP spoofing, port stealing
 - **Layer 3 (IP)**
 - exploiting ICMP or IGMP control protocols, dynamic host configuration protocols (BOOTP/DHCP/MobileIP), routing protocol (RIP and OSPF)
 - **Layer 7 (application)**
 - e.g. DNS poisoning

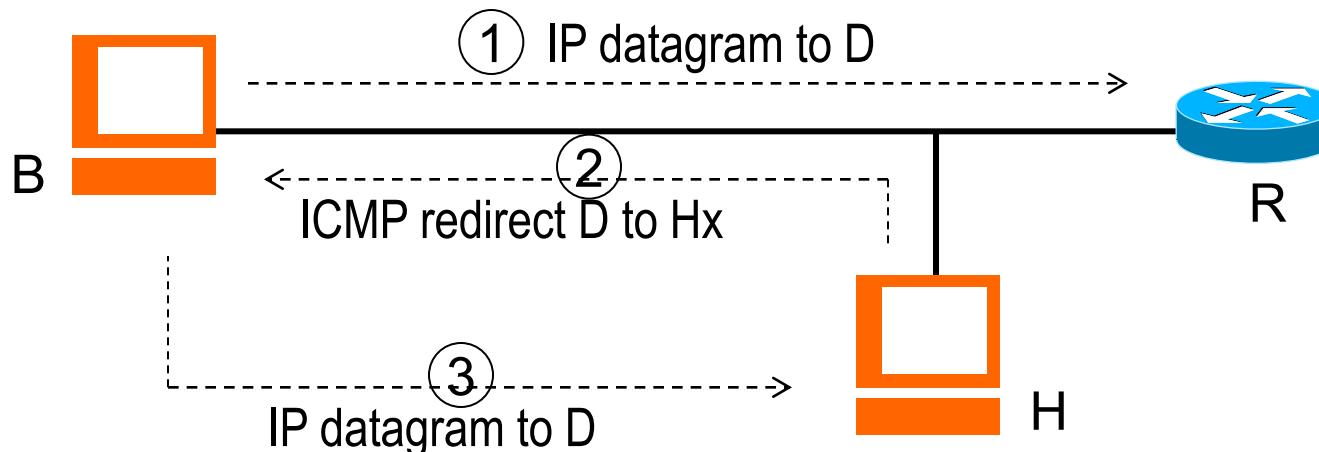


Routing attacks: ICMP Destination unreachable

- The attacker sends a spoofed *ICMP Destination unreachable* message to the victim in the same LAN
- Possible subtypes of *Destination unreachable*:
 - **Network unreachable**
 - **Host unreachable**
 - **Protocol unreachable**
 - **Port unreachable**
 - **Fragmentation needed but don't fragment bit set**
 - **Destination host unknown**
 - **Destination network unknown**
- Can be used to mount a DoS attack
 - e.g. in order to start a spoofing attack

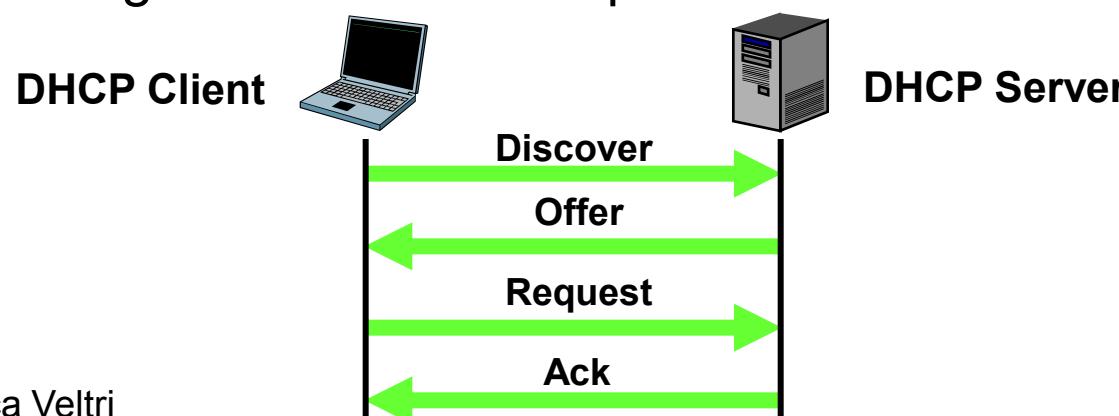
Routing attacks: ICMP Redirect

- The attacker sends a spoofed *ICMP Redirect* message to the victim in the same LAN pretending to be a better route for a destination
 - alternative respect to ARP spoofing
 - it uses standard ICMP message
 - all packets or connections to internet will be redirected to the new node which, in turn, may
 - drop them (DoS)
 - forward them to the real gateway (MITM)



Routing attacks: BOOTP/DHCP attack

- Spoofing attack that exploits BOOTP/DHCP configuration protocols
- The attack pretends to be a DHCP server and tries to win the race condition with the real one to force the client to accept the attacker's reply
- This way the attacker is able to manipulate the IP, DNS or router parameters and hijack all the outgoing traffic generated by the DHCP clients
- The resulting attack is a half-duplex MITM or DoS attack





Application protocol attacks

- Exploiting specific application protocols (DNS, Mail, FTP, etc.)
- Examples:
 - **email**
 - mail spamming
 - mail spoofing
 - mail phishing
 - **DNS**
 - pharming – an attack aiming to redirect a website's traffic to another, bogus website
 - "hosts" file poisoning
 - » changing the "hosts" file on a victim's computer
 - DNS spoofing
 - » returning a fake DNS record (either against a host or a DNS server)
 - DNS cache poisoning
 - » exploitation of a flaw in the DNS software that can make it accept incorrect information

Vulnerability scanning



Vulnerability scanning

● Types of vulnerability scanners

➤ **network and port scanner**

- tells which hosts and network services are running and reachable
 - e.g. Nmap

➤ **network vulnerability scanner**

- looks for possible vulnerabilities associated to detected running application
 - e.g. Nessus, SAINT, OpenVAS

➤ **web application security scanner**

- automated crawling and testing of web applications
 - e.g. Indusface WAS, ScanMyServer, SUCURI, SSL Labs, Quttera, Detectify, Siteguarding, etc
 - https://owasp.org/www-community/Vulnerability_Scanning_Tools

➤ **database security scanner**

➤ **host based vulnerability scanner**

➤ **single vulnerability test**



Network scanning

- Also referred as network enumerating
- Activity through specific software for scanning a network for discovering active hosts and open ports (Port scanning)
 - e.g. ping, traceroute, nmap
- This is often used by administrators to check the security of their networks
- Can be used by hackers to identify running services on a host with the view to compromising it
- Related to network scanning are:
 - Vulnerability scanning
 - Penetration test



Network scanning (cont.)

- What a network scanner can do

- **Host discovery**

- identifying hosts on a network, for example listing the hosts which respond to pings, or which have a particular port open

- **Port scanning**

- enumerating the open ports on one or more target hosts

- **Service version detection**

- interrogating listening network services listening on remote devices to determine the application name and version number

- **OS detection**

- remotely determining the operating system and some hardware characteristics of network devices



Host Discovery

- Host discovery may be performed by using different techniques
 - **ICMP scan**
 - determines if a host responds to ICMP requests, such as echo (ping), netmask, etc
 - **TCP SYN/ACK/RST**
 - tries to open a TCP connection using port numbers
 - associated to most common application services (e.g. HTTPd), or
 - by a given set
 - two modes (see later):
 - user space (TCP connect())
 - root space (TCP SYN/ACK/RST)



Host Discovery: ICMP Echo Request

- Host discovery through ICMP Echo Request (pingscan):
 - ICMP echo datagrams are sent to a given host or to all hosts in a subnetwork
 - The attacker collects the replies and determines which hosts are actually alive
- Example:

Starting nmap V. 2.12 by Fyodor (www.insecure.org/nmap/)

Host cisco-sales.ns.com (195.121.31.11) appears to be up.

Host sales1.ns.com (195.121.31.19) appears to be up.

Host sales4.ns.com (195.121.31.22) appears to be up.

Host sales2.ns.com (195.121.31.43) appears to be up.

Host sales3.ns.com (195.121.31.181) appears to be up.

Nmap run completed -- 256 IP addresses (5 hosts up) scanned in 1 second



Port scanning

- A port scanner may scan
 - **the most common port numbers, or**
 - **ports most commonly associated with vulnerable services, on a given host, or**
 - **a given list of TCP and UDP port numbers**
- The result of a scan on a target {host,proto,port} can be:
 - **Open or Accepted**
 - the host sent a reply indicating that a service is listening on the port
 - open ports may reveal vulnerabilities associated with
 - the program responsible for delivering the service
 - the operating system that is running on the host
 - **Closed or Denied or Not Listening**
 - the host sent a reply indicating that connections will be denied to the port
 - **Filtered, Dropped or Blocked**
 - there was no reply from the host



Port scanning (cont.)

- Port scanning types:

- **TCP scanning**

- the simplest port scanners
 - use the operating system's network functions (e.g. connect() call)
 - if a port is open, the operating system completes the TCP three-way handshake, and the port scanner immediately closes the connection; otherwise an error code is returned
 - has the advantage that the user does not require special privileges
 - however, it prevents low-level control

- **SYN scanning**

- the port scanner generates raw IP packets itself (TCP SYN packet), and monitors for responses (TCP SYN-ACK packet), rather than use the operating system's network functions
 - the scanner responds with a TCP RST packet, closing the connection before the handshake is completed
 - also known as "half-open scanning", because it never actually opens a full TCP connection



Port scanning (cont.)

- Port scanning types: (cont.)

- **ACK scanning**

- it does not exactly determine whether the port is open or closed, but whether the port is filtered or unfiltered

- **UDP scanning**

- if a UDP packet is sent to a port that is not open, some systems respond with an ICMP port unreachable message
 - an alternative approach is to send application-specific UDP packets, hoping to generate an application layer response

- Note: there is debate over which scan is less intrusive on the target host between TCP connect or TCP SYN scans

- SYN scan has the advantage that the individual services never actually receive a connection; however, the RST during the handshake can cause problems for some network stacks, in particular simple devices, like printers or other smart objects
 - connect scan uses more resources and may cause some services to crash



Network vulnerability scanners

- They usually combine a network scanner with a DB of known application weaknesses and vulnerabilities
 - **for each vulnerability they usually associate:**
 - degree of risk
 - recommendations on how to mitigate it
- Examples of vulnerability scanning tools:
 - **Nessus**
 - **OpenVAS**



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Software vulnerabilities

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://www.tlc.unipr.it/veltri>



Software vulnerabilities

- Vulnerabilities in application, utility, or operating system code
- Software vulnerabilities are often caused by a bug or weakness present in the software
 - **application implementations**
 - **operating systems**
- Many computer security vulnerabilities result from poor programming practices

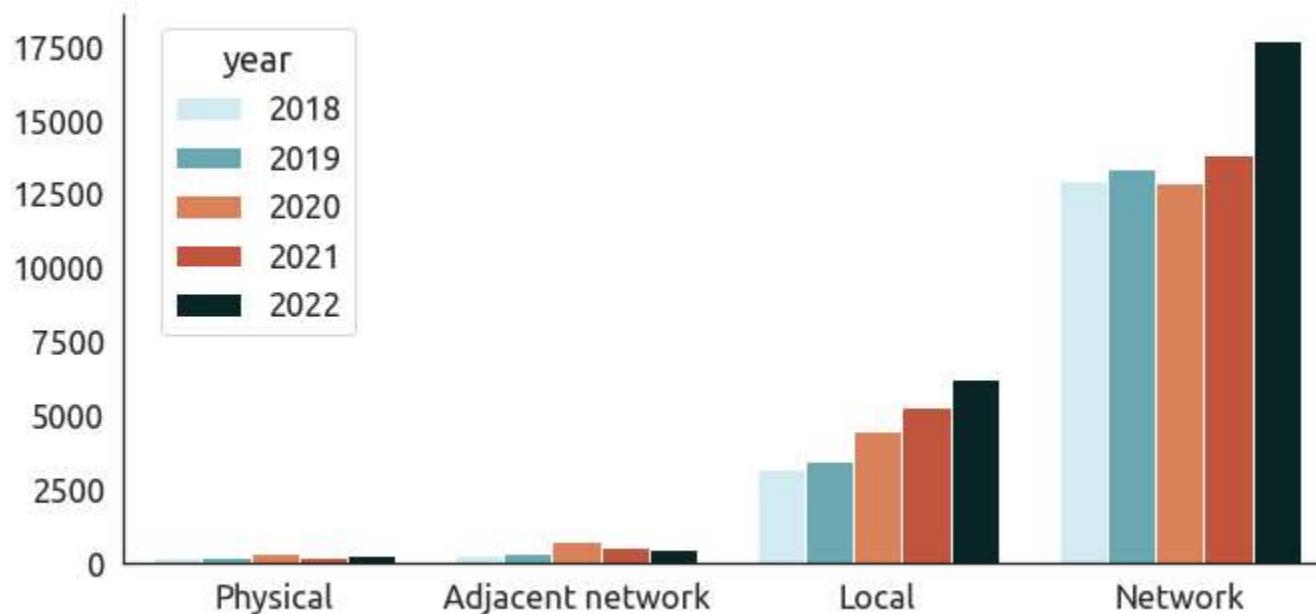


2022 CWE Most Dangerous Software Weaknesses

- 1 CWE-787 Out-of-bounds Write
- 2 CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- 3 CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- 4 CWE-20 Improper Input Validation
- 5 CWE-125 Out-of-bounds Read
- 6 CWE-78 Improper neutralization of special elements used in an OS command (OS Command Injection')
- 7 CWE-416 Use After Free
- 8 CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- 9 CWE-352 Cross-Site Request Forgery (CSRF)
- 10 CWE-434 Unrestricted Upload of File with Dangerous Type
- 11 CWE-476 NULL Pointer Dereference
- 12 CWE-502 Deserialization of Untrusted Data
- 13 CWE-190 Integer Overflow or Wraparound
- 14 CWE-287 Improper Authentication
- 15 CWE-798 Use of Hard-coded Credentials
- 16 CWE-862 Missing Authorization
- 17 CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')
- 18 CWE-306 Missing Authentication for Critical Function
- 19 CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer
- 20 CWE-276 Incorrect Default Permissions
- 21 CWE-918 Server-Side Request Forgery (SSRF)
- 22 CWE-362 Concurrent execution using shared resource with improper synchronization (Race condition)
- 23 CWE-400 Uncontrolled Resource Consumption
- 24 CWE-611 Improper Restriction of XML External Entity Reference
- 25 CWE-94 Improper Control of Generation of Code ('Code Injection')

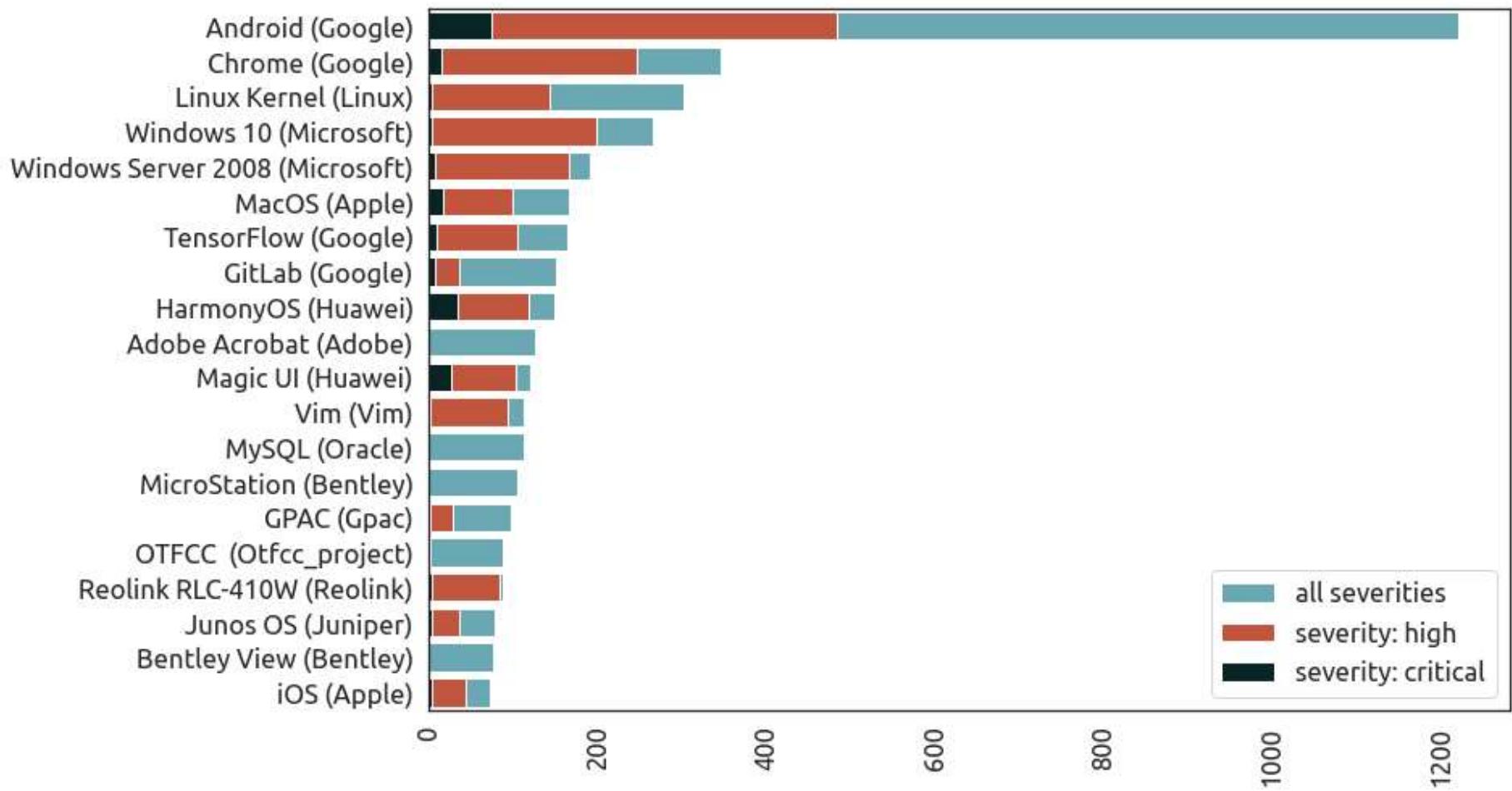
CVE analysis

- Attack vectors^(*):



CVE analysis (cont.)

- The top 20 products with the most CVEs in 2022:





Top 10 web application vulnerabilities (2021)

- Open Web Application Security Project (OWASP) Top 10 vulnerabilities
 - 1) **Broken Access Control**
 - 2) **Cryptographic Failures**
 - 3) **Injection**
 - 4) **Insecure Design**
 - 5) **Security Misconfiguration**
 - 6) **Vulnerable and Outdated Components**
 - 7) **Identification and Authentication Failures**
 - 8) **Software and Data Integrity Failures**
 - 9) **Security Logging and Monitoring Failures**
 - 10) **Server Side Request Forgery (SSRF)**



Software Error Categories

- Possible classification CWE/SANS Most Dangerous Software Errors (2011):

- **1) Insecure Interaction Between Components**

- Improper Neutralization of Special Elements used in an SQL Command (“SQL Injection”)
 - Improper Neutralization of Special Elements used in an OS Command (“OS Command Injection”)
 - Improper Neutralization of Input During Web Page Generation (“Cross-site Scripting”)
 - Unrestricted Upload of File with Dangerous Type
 - Cross-Site Request Forgery (CSRF)
 - URL Redirection to Untrusted Site (“Open Redirect”)



Software Error Categories (cont.)

- Classification (cont.):

- **2) Risky Resource Management**

- Buffer Copy without Checking Size of Input (“Classic Buffer Overflow”)
 - Improper Limitation of a Pathname to a Restricted Directory (“Path Traversal”)
 - Download of Code Without Integrity Check
 - Inclusion of Functionality from Untrusted Control Sphere
 - Use of Potentially Dangerous Function
 - Incorrect Calculation of Buffer Size
 - Uncontrolled Format String
 - Integer Overflow or Wraparound



Software Error Categories (cont.)

- Classification (cont.):

- **3) Porous Defenses**

- Missing Authentication for Critical Function
 - Missing Authorization
 - Use of Hard-coded Credentials
 - Missing Encryption of Sensitive Data
 - Reliance on Untrusted Inputs in a Security Decision
 - Execution with Unnecessary Privileges
 - Incorrect Authorization
 - Incorrect Permission Assignment for Critical Resource
 - Use of a Broken or Risky Cryptographic Algorithm
 - Improper Restriction of Excessive Authentication Attempts
 - Use of a One-Way Hash without a Salt



Software Error Categories (cont.)

- While type 3 (Porous Defense) vulnerabilities are mainly related to poor implementation of protection mechanisms, type 1 and 2 vulnerabilities mainly refer to errors/weaknesses due to improper handling of program inputs
- These can mainly due to two different levels:
 - **errors/weaknesses at programming language level**
 - how programs are written and how they are translated into machine code
 - issues often related on the size of input data
 - » e.g. buffer overflow
 - **errors/weaknesses at program logic level**
 - how inputs are interpreted and handled by the logic of the program
 - incorrect handle of remote user inputs that makes the program to perform other actions
 - often due to execution of action based on user input values without a proper check of correctness (validation) of the input
 - » Injection attacks



Buffer Overflow

- Also known as a buffer overrun
- Very common attack mechanism
 - First widely used by the Morris Worm in 1988
- Prevention techniques known
- Still of major concern
 - Legacy of buggy code in widely deployed operating systems and applications
 - Continued careless programming practices by programmers
- Defined in the NIST Glossary of Key Information Security Terms as follows:
 - A condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system



Buffer Overflow Basics

- Programming error when a process attempts to store data beyond the limits of a fixed-sized buffer
- Overwrites adjacent memory locations
 - Locations could hold other program variables, parameters, or program control flow data
- Buffer could be located on the stack, in the heap, or in the data section of the process
- Consequences:
 - Corruption of program data
 - Unexpected transfer of control
 - Memory access violations
 - Execution of code chosen by attacker



Buffer overflow example

- Simple example of buffer overflow C code

```
#define FALSE 0
#define TRUE 1

void get_password(char* dest) { //get a password from a DB
    strcpy(dest, "SECRET");
}

int main(int argc, char *argv[]) {
    int valid= FALSE;
    char str1[8];
    char str2[8];
    printf("insert valid password: ");
    get_password(str1); //read the password from the user
    gets(str2);
    if (strncmp(str1,str2,8)==0) valid= TRUE; else valid= FALSE;
    printf("str1: %s\nstr2: %s\nvalid: %d\n", str1, str2,
    valid);
}
```



Buffer overflow example (cont.)

```
insert password: test
str1: SECRET
str2: test
valid: 0
```

```
insert password: 123456789abcdefghijklmnopqrstuvwxyz
str1: 9abcdefghijklmnopqrstuvwxyz
str2: 123456789abcdefghijklmnopqrstuvwxyz
valid: 0
```

```
insert password: aaaabbbaaaabbbbccccdd
str1: aaaabbbaaaabbbbccccdd
str2: aaaabbbaaaabbbbccccdd
valid: 1
```

```
insert password: badinputbadinput
str1: badinput
str2: badinputbadinput
valid: 1
```



Buffer overflow example (cont.)

Memory Address	Before gets(str2)	After gets(str2)	Contains value of
bffffbf4	34fcffbf 4 . . .	34fcffbf 3 . . .	argv
bffffbf0	01000000	01000000	argc
bffffbec	c6bd0340 . . . @	c6bd0340 . . . @	return addr
bffffbe8	08fcffbf	08fcffbf	old base ptr
bffffbe4	00000000	01000000	valid
bffffbe0	80640140 . d . @	00640140 . d . @	
bffffbdc	54001540 T . . @	4e505554 N P U T	str1[4-7]
bffffbd8	53544152 S T A R	42414449 B A D I	str1[0-3]
bffffbd4	00850408	4e505554 N P U T	str2[4-7]
bffffbd0	30561540 0 V . @	42414449 B A D I	str2[0-3]
.	



Buffer overflow attacks

- To exploit a buffer overflow an attacker needs:
 - **To identify a buffer overflow vulnerability in some program that can be triggered using externally sourced data under the attacker's control**
 - **To understand how that buffer is stored in memory and determine potential for corruption**
- Identifying vulnerable programs can be done by:
 - **Inspection of program source**
 - **Tracing the execution of programs as they process oversized input**
 - **Using tools such as fuzzing to automatically identify potentially vulnerable programs**
- What the attacker does with the resulting corruption of memory varies considerably, depending on what values are being overwritten

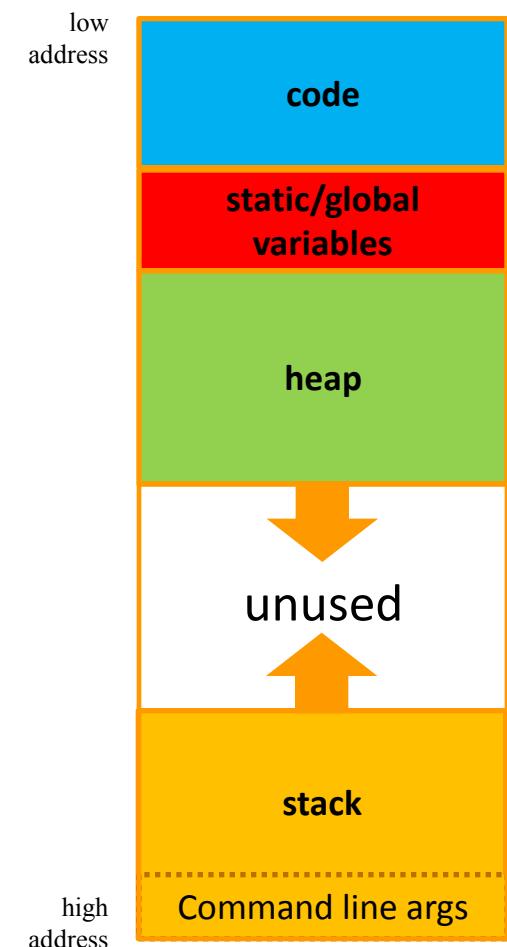


Stack Buffer Overflows

- Occur when buffer is located on stack
 - **also referred to as stack smashing**
 - **used by Morris Worm**
 - **exploits include an unchecked buffer overflow**
- Are still being widely exploited
- Stack frame
 - **When one function calls another it needs somewhere to save the return address**
 - **Also needs locations to save the parameters to be passed in to the called function and to possibly save register values**

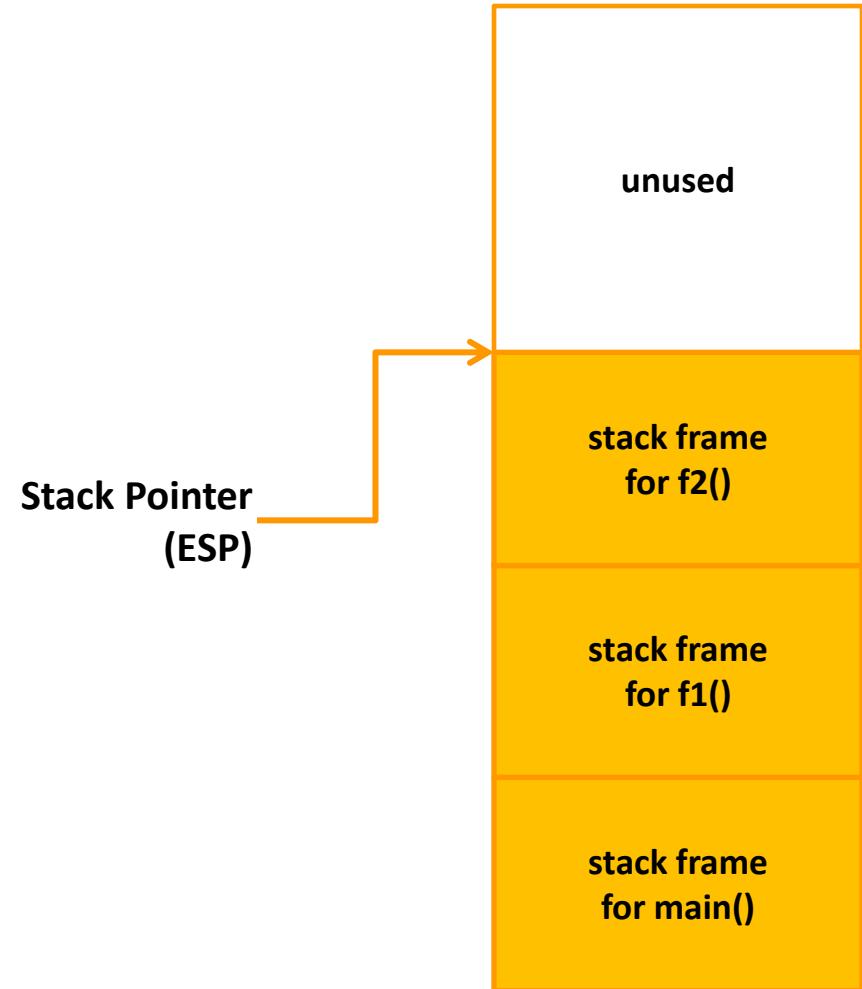
Program memory allocation

- Memory is allocated for each process (a running program) to store data and code
- This allocated memory consists of different segments:
 - **stack: for local variables**
 - **heap: for dynamic memory**
 - **data segment:**
 - global uninitialized variables
 - global initialized variables
 - **code segment**



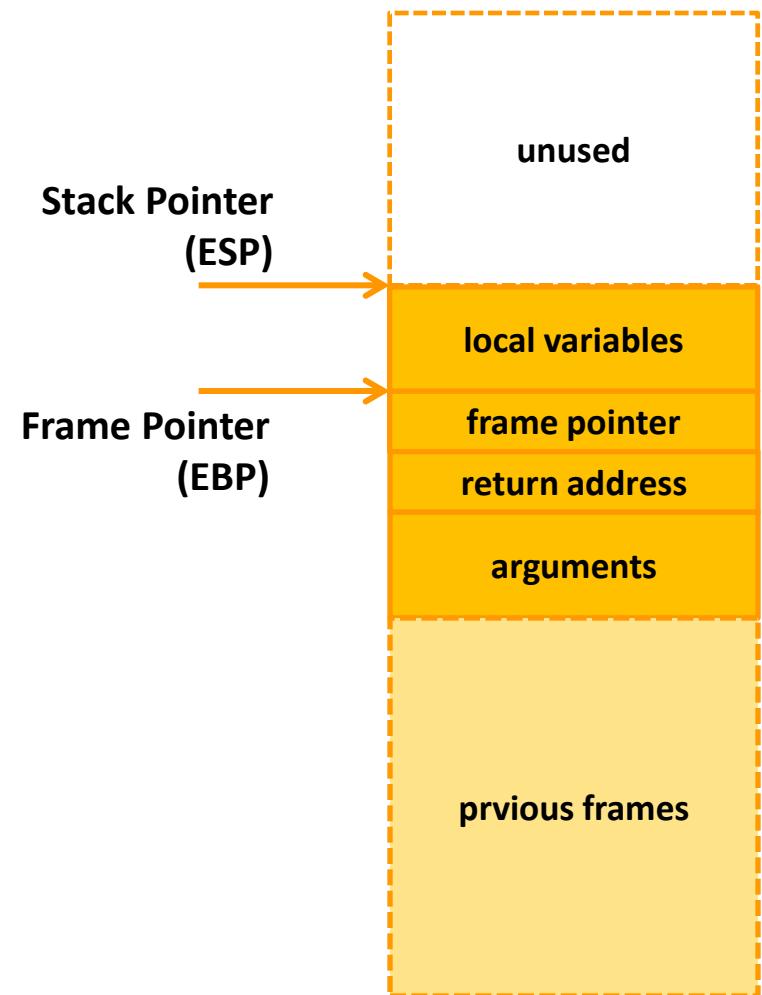
The stack

- The precise structure and organization of the stack depends on system architecture, operating system, and compilers we are using
- Typically, the stack grows downward
- The stack pointer (SP) refers to the last element on the stack
 - **on x86 architectures, the stack pointer is stored in the ESP (Extended Stack Pointer) register**



x86 stack frame

- In x86 architecture, each stack frame contains:
 - Function arguments
 - Local variables
 - Copies of registries that must be restored:
 - return address
 - previous frame pointer
- Frame pointer, named Extended Base Pointer (EBP), provides a starting point to local variables



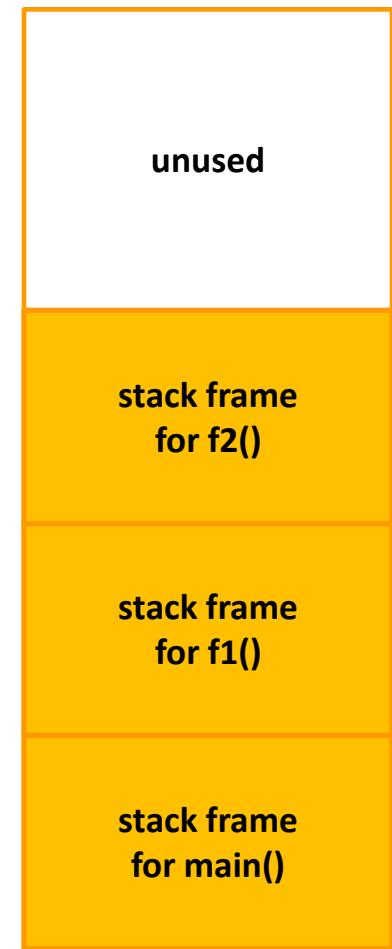
The stack (cont.)

- The stack consists of a sequence of *stack frames* (or activation records), each for each function call:

- **allocated on call**
- **de-allocated on return**

- In the stack example:

- **main() called f1()**
- **f1() called f2()**
- **f1() and f2() can be also the same function**
 - in case of recursion

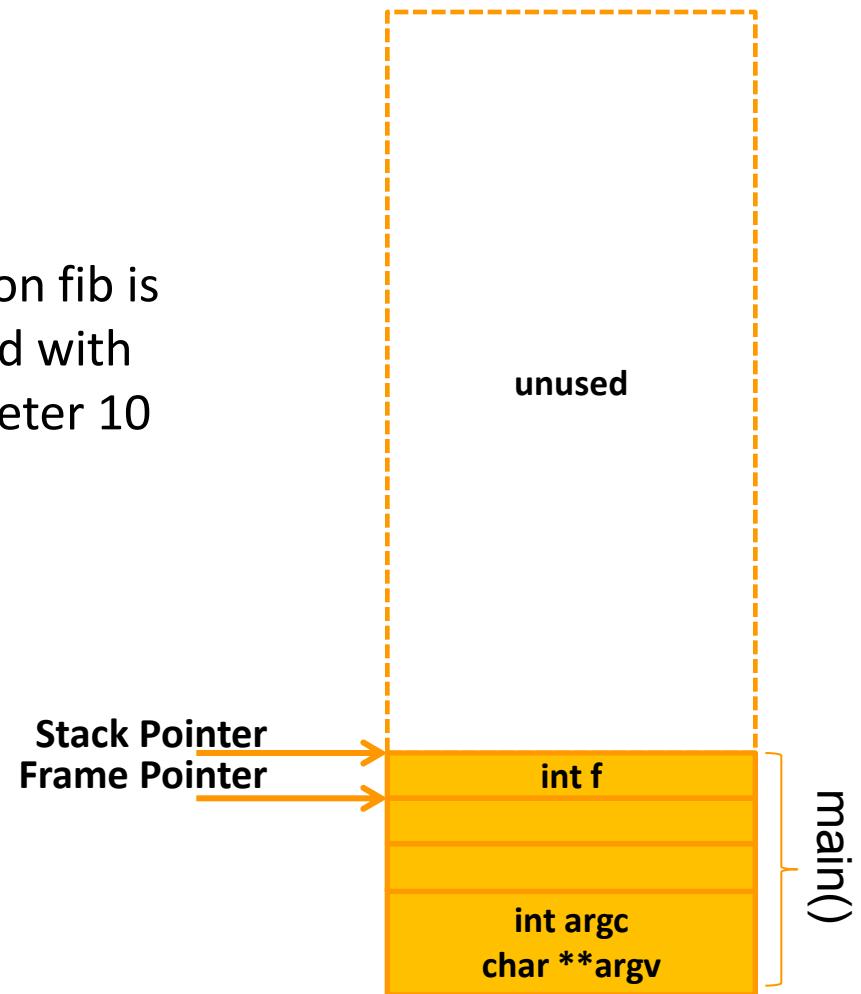


Stack frame: example

```
#include<cstdio>

int fibonacci(int n) {
    int f1, f2;
    if (n<=2) return 1;
    else {
        f1= fibonacci(n-1);
        f2= fibonacci(n-2);
        return f1+f2;
    }
}
int main () {
    int f= fibonacci(10);  
    printf("fib(10)= %d\n",f);
}
```

Function fib is
invoked with
parameter 10

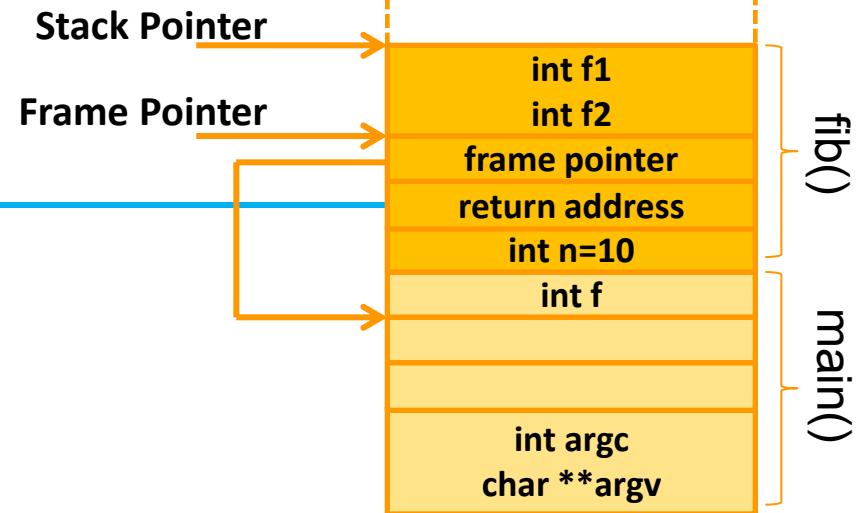


Stack frame: example

```
#include<cstdio>

int fibonacci(int n) {
    int f1, f2;
    if (n<=2) return 1;
    else {
        f1= fibonacci(n-1);
        f2= fibonacci(n-2);
        return f1+f2;
    }
}
int main () {
    int f= fibonacci(10);
    printf("fib(10)= %d\n",f);
}
```

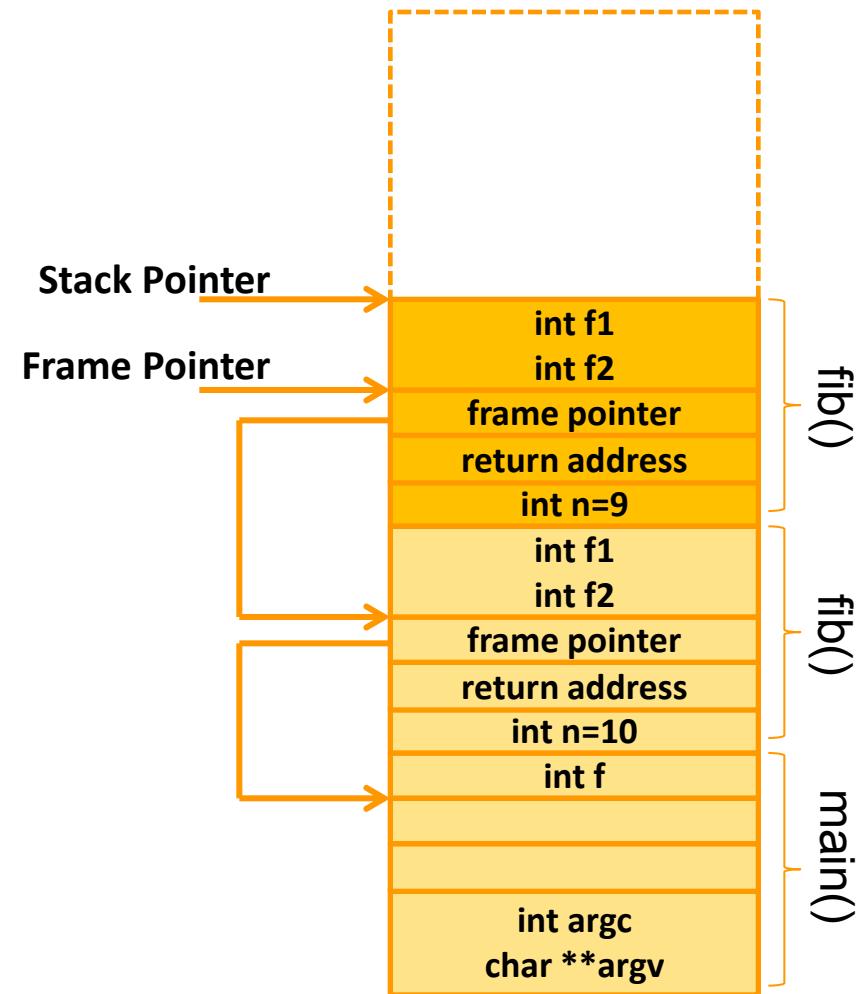
Stack frame is allocated and pointers updated



Stack frame: example

```
#include<cstdio>

int fibonacci(int n) {
    int f1, f2;
    if (n<=2) return 1;
    else {
        f1= fibonacci(n-1);
        f2= fibonacci(n-2);
        return f1+f2;
    }
}
int main () {
    int f= fibonacci(10);
    printf("fib(10)= %d\n",f);
}
```



Stack frame: example

```
#include<cstdio>

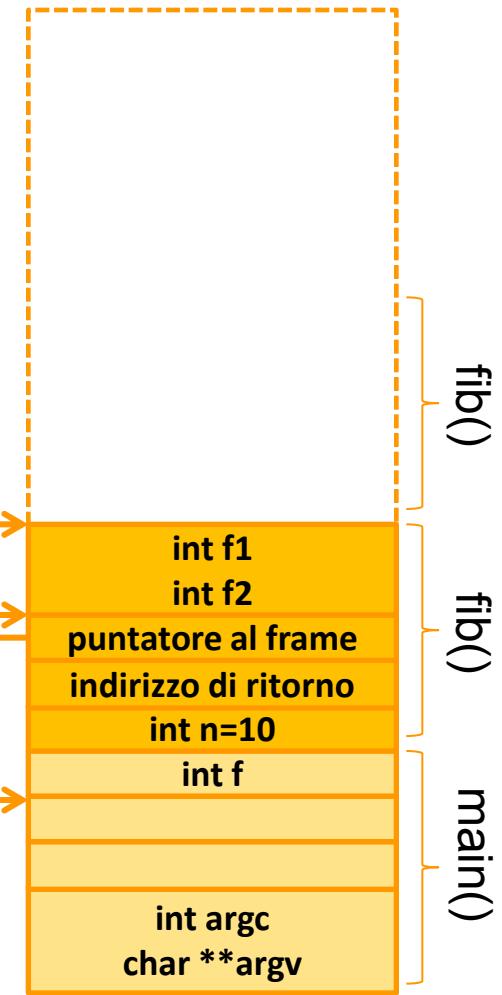
int fibonacci(int n) {
    int f1, f2;
    if (n<=2) return 1;
    else {
        f1= fibonacci(n-1);
        f2= fibonacci(n-2);
        return f1+f2;
    }
}
int main () {
    int f= fibonacci(10);
    printf("fib(10)= %d\n",f);
}
```

When a function returns, pointers are updated

Function result (if any) is copied in a register

Stack Pointer

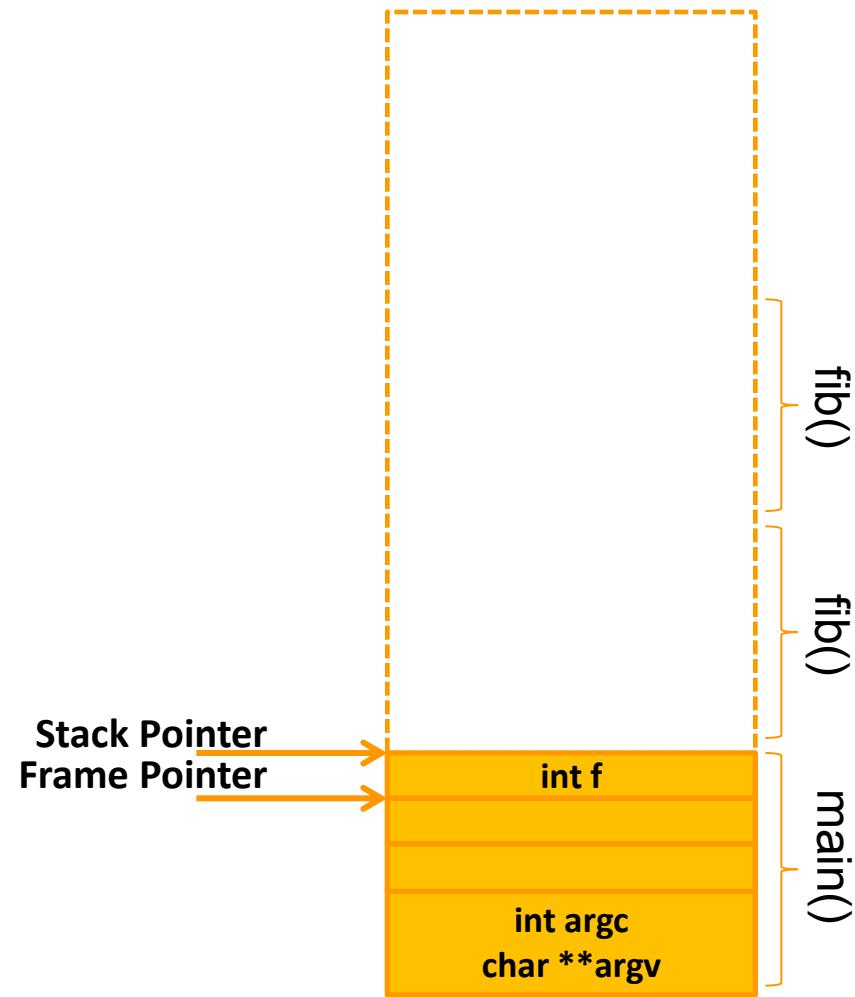
Frame Pointer



Stack frame: example

```
#include<cstdio>

int fibonacci(int n) {
    int f1, f2;
    if (n<=2) return 1;
    else {
        f1= fibonacci(n-1);
        f2= fibonacci(n-2);
        return f1+f2;
    }
}
int main () {
    int f= fibonacci(10);
    printf("fib(10)= %d\n",f);
}
```





The heap

- Memory allocation and de-allocation in the stack is very fast
 - **However, this memory cannot be used after a function returns**
- The heap is used to store dynamically allocated data that outlive function calls:
 - **This area is under programmer's responsibility**



Simple Stack Overflow Example

```
void hello(char *tag)
{
    char inp[16];

    printf("Enter value for %s: ", tag);
    gets(inp);
    printf("Hello your %s is %s\n", tag, inp);
}
```

(a) Basic stack overflow C code

```
$ cc -g -o buffer2 buffer2.c

$ ./buffer2
Enter value for name: Bill and Lawrie
Hello your name is Bill and Lawrie
buffer2 done

$ ./buffer2
Enter value for name: XXXXXXXXXXXXXXXXXXXXXXXXX
Segmentation fault (core dumped)

$ perl -e 'print pack("H*", "414243444546474851525354555657586162636465666768
e8fffffb948304080a4e4e4e4e0a");' | ./buffer2
Enter value for name:
Hello your Re?pyjuEA is ABCDEFGHQRSTUVWXabcdefguyu
Enter value for Kyyu:
Hello your Kyyu is NNNN
Segmentation fault (core dumped)
```

(b) Basic stack overflow example runs



Simple Stack Overflow Stack Values

Memory Address	Before gets(inp)	After gets(inp)	Contains value of
bffffbe0	3e850408 > . . .	00850408 . . .	tag
bffffbdc	f0830408 . . .	94830408 . . .	return addr
bffffbd8	e8fbffbf . . .	e8ffffbf . . .	old base ptr
bffffbd4	60840408 ` . . .	65666768 e f g h	
bffffbd0	30561540 0 V . @	61626364 a b c d	
bffffbcc	1b840408 . . .	55565758 U V W X	inp[12-15]
bffffbc8	e8fbffbf . . .	51525354 Q R S T	inp[8-11]
bffffbc4	3cfcffbf < . . .	45464748 E F G H	inp[4-7]
bffffbc0	34fcffbf 4 . . .	41424344 A B C D	inp[0-3]
.	



Shellcode

- Code supplied by attacker
 - Often saved in buffer being overflowed
 - Traditionally transferred control to a user command-line interpreter (shell)
- Machine code
 - Specific to processor and operating system
 - Traditionally needed good assembly language skills to create
 - More recently a number of sites and tools have been developed that automate this process
 - e.g. Metasploit Project - it provides useful information to people who perform penetration, IDS signature development, and exploit research



Example UNIX Shellcode

```
int main (int argc, char *argv[])
{
    char *sh;
    char *args[2];

    sh = "/bin/sh";
    args[0] = sh;
    args[1] = NULL;
    execve (sh, args, NULL);
}
```

(a) Desired shellcode code in C

```
nop                                //end of nop sled
nop
jmp find                            //jump to end of code
cont: pop %esi                      //pop address of sh off stack into %esi
    xor %eax, %eax                  //zero contents of EAX
    mov $al, 0x7(%esi)              //copy zero byte to end of string sh (%esi)
    lea (%esi), %ebx                //load address of sh (%esi) into %ebx
    mov %ebx,%08(%esi)              //save address of sh in args [0] (%esi+8)
    mov %eax,0xc(%esi)              //copy zero to args[1] (%esi+c)
    mov $0xb,%al                   //copy execve syscall number (11) to AL
    mov %esi,%ebx                  //copy address of sh (%esi) into %ebx
    lea 0x8(%esi),%ecx              //copy address of args [0] (%esi+8) to %ecx
    lea 0xc(%esi),%edx              //copy address of args[1] (%esi+c) to %edx
    int $0x80                        //software interrupt to execute syscall
find: call cont                     //call cont which saves next address on stack
sh: .string "/bin/sh"                //string constant
args: .long 0                        //space used for args array
    .long 0                          //args[1] and also NULL for env array
```

(b) Equivalent position-independent x86 assembly code

```
90 90 eb 1a 5e 31 c0 88 46 07 8d 1e 89 5e 08 89
46 0c b0 0b 89 f3 8d 4e 08 8d 56 0c cd 90 e8 e1
ff ff ff 2f 62 69 6e 2f 73 68 20 20 20 20 20 20
```

(c) Hexadecimal values for compiled x86 machine code



Buffer Overflow Defenses

- Buffer overflows are widely exploited
- Two broad defense approaches
 - **Compile-time**
 - Aim to harden programs to resist attacks in new programs
 - **Run-time**
 - Aim to detect and abort attacks in existing programs



Compile-Time Defenses: Programming Language

- Use a modern high-level language
 - Not vulnerable to buffer overflow attacks
 - Compiler enforces range checks and permissible operations on variables
- Disadvantages
 - Additional code must be executed at run time to impose checks
 - Flexibility and safety comes at a cost in resource use
 - Distance from the underlying machine language and architecture means that access to some instructions and hardware resources is lost
 - Limits their usefulness in writing code, such as device drivers, that must interact with such resources



Compile-Time Defenses: Safe Coding Techniques

- C designers placed much more emphasis on space efficiency and performance considerations than on type safety
 - **Assumed programmers would exercise due care in writing code**
- Programmers need to inspect the code and rewrite any unsafe coding
 - **An example of this is the Open BSD project**
- Programmers have audited the existing code base, including the operating system, standard libraries, and common utilities
 - **This has resulted in what is widely regarded as one of the safest operating systems in widespread use**



Examples of Unsafe C Code

```
int copy_buf(char *to, int pos, char *from, int len)
{
    int i;
    for (i=0; i<len; i++) {
        to[pos] = from[i];
        pos++;
    }
    return pos;
}
```

(a) Unsafe byte copy

```
short read_chunk(FILE fil, char *to)
{
    short len;
    fread(&len, 2, 1, fil);           /* read length of binary data */
    fread(to, 1, len, fil);          /* read len bytes of binary data */
    return len;
}
```

(b) Unsafe byte input



Compile-Time Defenses: Language Extensions/Safe Libraries

- Handling dynamically allocated memory is more problematic because the size information is not available at compile time
 - **Requires an extension and the use of library routines**
 - Programs and libraries need to be recompiled
 - Likely to have problems with third-party applications
- Concern with C is use of unsafe standard library routines
 - **One approach has been to replace these with safer variants**
 - Libsafe is an example
 - Library is implemented as a dynamic library arranged to load before the existing standard libraries



Compile-Time Defenses: Stack Protection

- Add function entry and exit code to check stack for signs of corruption
- Use random canary
 - **value needs to be unpredictable**
- Stackshield and Return Address Defender (RAD)
 - **GCC extensions that include additional function entry and exit code**
 - function entry writes a copy of the return address to a safe region of memory
 - function exit code checks the return address in the stack frame against the saved copy
 - if change is found, aborts the program



Run-Time Defenses: Executable Address Space Protection

- Use virtual memory support to make some regions of memory non-executable
 - Requires support from memory management unit (MMU)
 - Long existed on SPARC / Solaris systems
 - Recent on x86 Linux/Unix/Windows systems
- Issues
 - Support for executable stack code
 - Special provisions are needed



Run-Time Defenses: Address Space Randomization

- Manipulate location of key data structures
 - stack, heap, global data
 - using random shift for each process
 - large address range on modern systems means wasting some has negligible impact
- Randomize location of heap buffers
- Random location of standard library functions



Replacement Stack Frame

- Variant that overwrites buffer and saved frame pointer address
 - **Saved frame pointer value is changed to refer to a dummy stack frame**
 - **Current function returns to the replacement dummy frame**
 - **Control is transferred to the shellcode in the overwritten buffer**
- Off-by-one attacks
 - **Coding error that allows one more byte to be copied than there is space available**
- Defenses
 - **Any stack protection mechanisms to detect modifications to the stack frame or return address by function exit code**
 - **Use non-executable stacks**
 - **Randomization of the stack in memory and of system libraries**



Return to System Call

- Stack overflow variant replaces return address with standard library function
 - Response to non-executable stack defenses
 - Attacker constructs suitable parameters on stack above return address
 - Function returns and library function executes
 - Attacker may need exact buffer address
 - Can even chain two library calls
- Defenses
 - Any stack protection mechanisms to detect modifications to the stack frame or return address by function exit code
 - Use non-executable stacks
 - Randomization of the stack in memory and of system libraries



Heap Overflow

- Attack buffer located in heap
 - Typically located above program code
 - Memory is requested by programs to use in dynamic data structures (such as linked lists of records)
- No return address
 - Hence no easy transfer of control
 - May have function pointers can exploit
 - Or manipulate management data structures
- Defenses
 - Making the heap non-executable
 - Randomizing the allocation of memory on the heap



Example Heap Overflow Attack

```
/* record type to allocate on heap */
typedef struct chunk {
    char inp[64];           /* vulnerable input buffer */
    void (*process)(char *); /* pointer to function to process inp */
} chunk_t;

void showlen(char *buf)
{
    int len;
    len = strlen(buf);
    printf("buffer5 read %d chars\n", len);
}

int main(int argc, char *argv[])
{
    chunk_t *next;

    setbuf(stdin, NULL);
    next = malloc(sizeof(chunk_t));
    next->process = showlen;
    printf("Enter value: ");
    gets(next->inp);
    next->process(next->inp);
    printf("buffer5 done\n");
}
```

(a) Vulnerable heap overflow C code



Global Data Overflow

- Defenses
 - Non executable or random global data region
 - Move function pointers
 - Guard pages
- Can attack buffer located in global data
 - May be located above program code
 - If has function pointer and vulnerable buffer
 - Or adjacent process management tables
 - Aim to overwrite function pointer later called



Example Global Data Overflow Attack

```
/* global static data - will be targeted for attack */
struct chunk {
    char inp[64];           /* input buffer */
    void (*process)(char *); /* pointer to function to process it */
} chunk;

void showlen(char *buf)
{
    int len;
    len = strlen(buf);
    printf("buffer6 read %d chars\n", len);
}

int main(int argc, char *argv[])
{
    setbuf(stdin, NULL);
    chunk.process = showlen;
    printf("Enter value: ");
    gets(chunk.inp);
    chunk.process(chunk.inp);
    printf("buffer6 done\n");
}
```

(a) Vulnerable global data overflow C code



Injection Attacks

- Flaws relating to invalid handling of input data, specifically when program input data can accidentally or deliberately influence the flow of execution of the program
- Most common case when input data are passed as a parameter to another program on the system
- Most often occur in scripting languages
 - e.g. Perl, PHP, Python, sh, SQL
 - often used as Web CGI scripts



A Web CGI Injection Attack

```
1 #!/usr/bin/perl
2 # finger.cgi - finger CGI script using Perl5 CGI module
3
4 use CGI;
5 use CGI::Carp qw(fatalToBrowser);
6 $q = new CGI; # create query object
7
8 # display HTML header
9 print $q->header,
10 $q->start_html('Finger User'),
11 $q->h1('Finger User'),
12 print "<pre>";
13
14 # get name of user and display their finger details
15 $user = $q->param("user");
16 print `/usr/bin/finger -sh $user`;
17
18 # display HTML footer
19 print "</pre>";
20 print $q->end_html;
```

(a) Unsafe Perl finger CGI script

```
<html><head><title>Finger User</title></head><body>
<h1>Finger User</h1>
<form method=post action="finger.cgi">
<b>Username to finger</b>: <input type=text name=user value="">
<p><input type=submit value="Finger User">
</form></body></html>
```

(b) Finger form

Example of correct input:

lpb



Example of command injection:

xxx; echo attack success; ls -lfinger*



```
Finger User
Login Name    TTY  Idle  Login Time  Where
lpb  Lawrie Brown   p0  Sat 15:24  pppd1.grapevine
```

```
Finger User
attack success
-rwxr-xr-x  1 lpb  staff  537 Oct 21 16:19 finger.cgi
-rw-r--r--  1 lpb  staff  251 Oct 21 16:14 finger.html
```

(c) Expected and subverted finger CGI responses

```
14 # get name of user and display their finger details
15 $user = $q->param("user");
16 die "The specified user contains illegal characters!" unless ($user =~ /\w+/);
17 print `/usr/bin/finger -sh $user`;
```

(d) Safety extension to Perl finger CGI script



PHP/SQL Injection Example

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" . $name . "';";
$result = mysql_query($query);
```

(a) Vulnerable PHP code

```
$name = $_REQUEST['name'];
$query = "SELECT * FROM suppliers WHERE name = '" .
mysql_real_escape_string($name) . "';";
$result = mysql_query($query);
```

(b) Safer PHP code

Example of injection input:

Bob'; drop table suppliers



Cross Site Scripting (XSS) Attacks

- Attacks where input provided by one user is subsequently output to another user
- Commonly seen in scripted Web applications
 - **vulnerability involves the inclusion of script code in the HTML content**
 - **script code may need to access data associated with other pages**
 - **browsers impose security checks and restrict data access to pages originating from the same site**
- Exploit assumption that all content from one site is equally trusted and hence is permitted to interact with other content from the site
- XSS reflection vulnerability
 - **attacker includes the malicious script content in data supplied to a site**



Input Fuzzing

- Developed by Professor Barton Miller at the University of Wisconsin Madison in 1989
- Software testing technique that uses randomly generated data as inputs to a program
 - **range of inputs is very large**
 - **intent is to determine if the program or function correctly handles abnormal inputs**
 - **simple, free of assumptions, cheap**
 - **assists with reliability as well as security**
- Can also use templates to generate classes of known problem inputs
 - **disadvantage is that bugs triggered by other forms of input would be missed**
 - **combination of approaches is needed for reasonably comprehensive coverage of the inputs**



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Firewalls

Luca Veltri

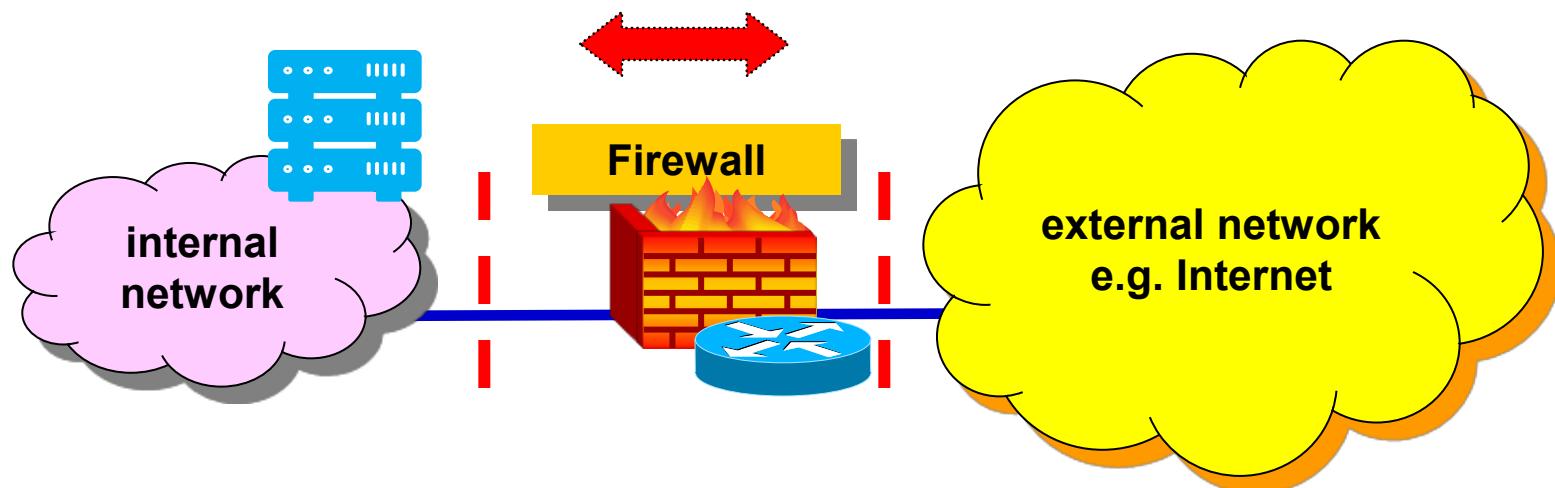
(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>

Firewall

- A firewall is security system that controls the incoming and outgoing network traffic
 - by analyzing the data packets and determining whether they should be allowed through or not, based on a rule set
 - builds a bridge between a system (an internal network or computer) and an external network
 - can be implemented in HW or SW





Examples of firewalls

- Many routers that pass data between networks contain firewall components
 - **packet filtering (or screening) router**
- Also intermediate systems working at network or application level may act as firewall
 - **NAT**
 - **Application Level Gateways (e.g. proxies)**
- Many personal computer OSs include software-based firewalls to protect against threats from the attached network and/or from the public Internet
 - **personal firewalls**
- Example: Linux iptables
 - may work as both personal firewall or router firewall



Firewall Classification

- There are different types of firewalls depending on where the communication is taking place, where the communication is intercepted and the state that is being traced
 - **Network layer or packet filters**
 - Level 3-4 switches
 - Screening routers
 - **Network address translation nodes**
 - NATs and NAPTs
 - **Application-layer firewalls**
 - Host-based application firewalls
 - Network-based application firewalls (proxies)
 - Single-interface bastion hosts
 - Dual-homed systems



When a firewall does not guarantee security

- In case of errors or firewall misconfigurations:
 - **the firewall does not correctly implement the defined security policies**
- The system where the firewall is installed is vulnerable
 - **due to bugs or misconfiguration of the OS or other applications**
- Presence of other paths between the external and internal networks, by-passing the firewall, e.g.
 - **wireless (WiFi) Access Points connected to the internal network, not protected by the firewall**
 - **internal computers/smartphones with 3/4G connections**
- Attacks that start from internal system
 - **malicious users**
 - **caused by compromised mobile computers (laptops, tablets, smartphones)**
 - **caused by software starting from removable disks / memory sticks**

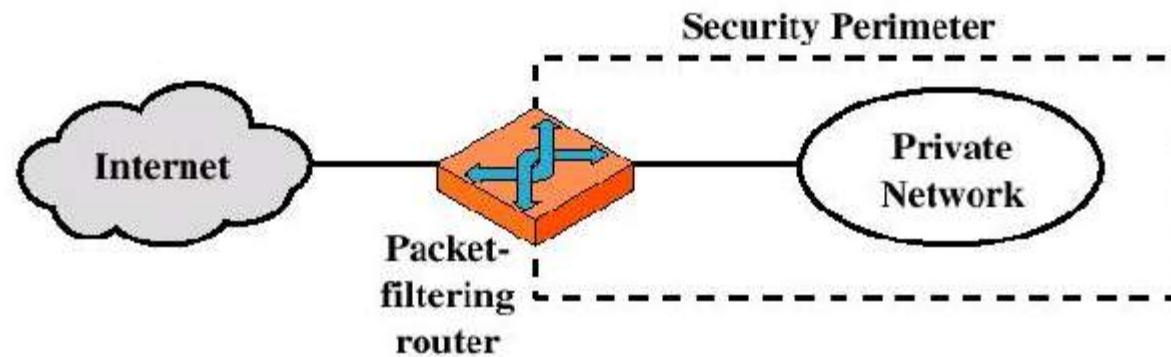
Packet Filters



Packet filters

- Also referred to as network layer firewalls
- They operate at a relatively low level of the protocol stack, not allowing single packets to pass through the firewall unless they match the established rule set
 - **can filter traffic based on many packet attributes like**
 - source/destination IP addresses,
 - transport protocol,
 - source/destination ports,
 - input/output interfaces,
 - many other packet header fields and attributes
 - **filter rules may be defined by a firewall administrator, or default rules may apply**

Packet Filtering Router



- IP router with filtering capabilities
 - also referred to as **Screening Router**
- Applies a set of rules to each incoming IP packet and then forwards or discards the packet
 - filter packets going in both directions
 - it is typically set up as a list of rules based on matches to fields in the IP and/or transport headers
 - two default policies (discard or forward)

Filtering rules: Example

- Filtering rules are listed in proper lists or tables, sometimes referred to as *Access control lists*, or *Chains*
- Example:
 - enabling traffic towards and from a mail server (**SMTP**) and web server (**HTTP**)

IP source	IP dest	Proto	Sorce port	Dest port	Action
*	160.78.1.1	tcp	> 1023	25	permit
*	160.78.1.2	tcp	> 1023	80	permit
160.78.1.0/24	*	*	*	*	permit
*	*	*	*	*	deny

Packet Filtering Operation



incoming packet

inspection of header and data

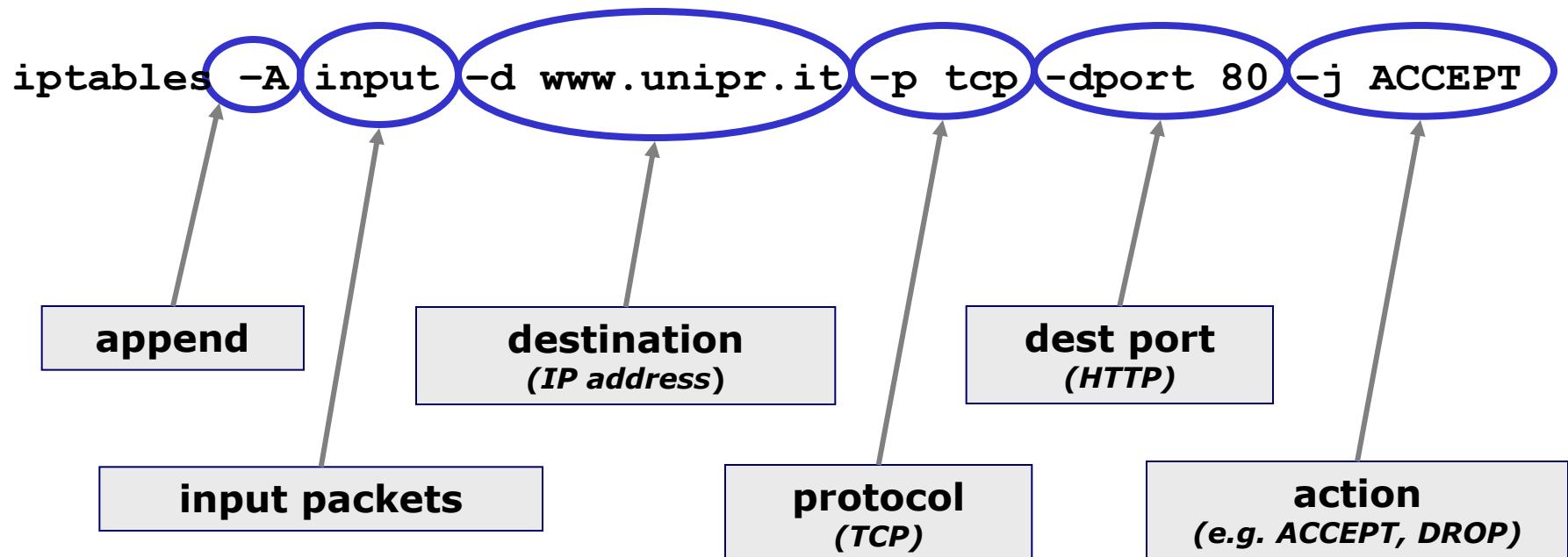
enforcing of filtering rules



access control lists

Actions: routing, drop, log, jump to an other list, others

Example of insertion of filtering rules





Stateless vs Stateful packet filters

- Generally fall into two sub-categories: stateless and stateful

- **Stateless packet filter**

- decision is per-packet based
 - no state related on previously processed packet
 - require less memory, and can be fast
 - cannot make complex decisions based on what stage communications between hosts have reached

- **Stateful packet filter**

- maintain context about active sessions, and use that "state information" to process packets
 - any existing communication is characterized by several properties (source and dest addresses, UDP/TCP ports, connection lifetime, etc.)
 - a firewall's state table is maintained and it contains state (connection) information relate to accepted packets
 - a packet matches an existing connection based on comparison with the state table
 - if a packet does not match an existing connection, it will be evaluated according to the ruleset for new connections



Packet Filters – Advantages/Disadvantages

- Advantages:
 - **Simplicity**
 - **High speed**
 - **Transparency to users**

- Disadvantages:
 - **Difficulty of setting up packet filter rules**
 - **Lack of authentication**

Linux Netfilter (iptables)

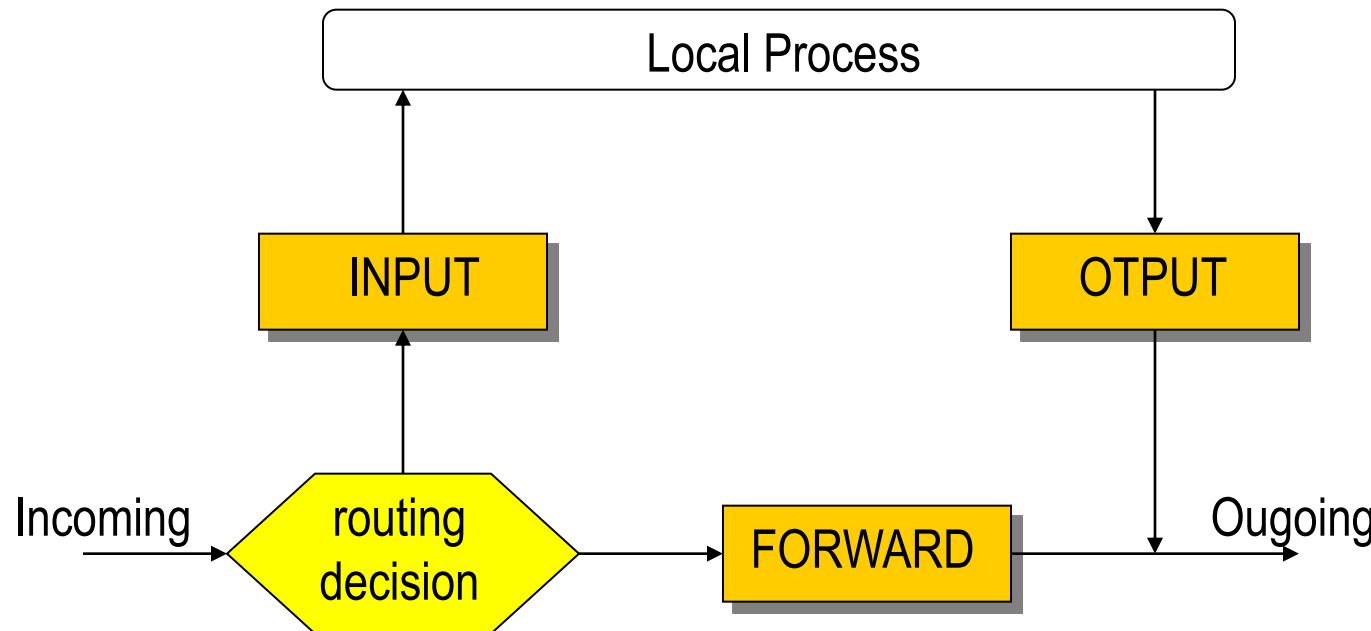


Linux packet filter

- Linux kernels have had packet filtering since the 1.1 series
- Packet filtering is implemented by netfilter
- Netfilter is a general framework inside the Linux kernel which other things can plug into
- The tool `iptables` talks to the kernel and tells it what packets to filter
 - **it inserts and deletes rules from the kernel's packet filtering table**
- Lists of filtering rules are called "chains"
 - **that are the Linux's access control lists**

Netfilter basic chains

- The kernel starts with three built-in lists of rules (chains) in the 'filter' table
 - **they are INPUT, OUTPUT and FORWARD**
 - **they can't be deleted**





Netfilter basic chains

- When a packet reaches a chain, that chain is examined to decide the fate of the packet
 - If the chain says to DROP the packet, it is killed there, but
 - if the chain says to ACCEPT the packet, it continues traversing the diagram
- A chain is a checklist of *rules*
 - each rule says `if the packet header looks like this, then here's what to do with the packet'
 - if the rule doesn't match the packet, then the next rule in the chain is consulted
 - finally, if there are no more rules to consult, then the kernel looks at the chain policy to decide what to do
 - in a security-conscious system, this policy usually tells the kernel to DROP the packet



iptables operations

- Operations to manage whole chains:
 - Create a new chain (-N)
 - Delete an empty chain (-X)
 - Change the policy for a built-in chain (-P)
 - List the rules in a chain (-L)
 - Flush the rules out of a chain (-F)
 - Zero the packet and byte counters on all rules in a chain (-Z)
- Operations to manipulate rules inside a chain:
 - Append a new rule to a chain (-A)
 - Insert a new rule at some position in a chain (-I)
 - Replace a rule at some position in a chain (-R)
 - Delete a rule at some position in a chain, or the first that matches (-D)



Managing an entire chain

- Creating a New Chain
 - **using the '-N' (or '--new-chain') command**
 - **e.g. `iptables -N test`**
- Deleting a Chain
 - **using the '-X' (or '--delete-chain') command**
 - **e.g. `iptables -X test`**
- Flushing a Chain
 - **using the '-F' (or '--flush') command**
 - **e.g. `iptables -F FORWARD`**
- Listing a Chain
 - **using the '-L' (or '--list') command**
 - '-n' (numeric) option prevents iptables from to lookup the IP addr
 - '-v' options shows you all the details of the rules



Managing an entire chain (cont.)

● Setting Policy

- the policy of the chain determines the default fate of the packet if no rule matches the packet
- only built-in chains (INPUT, OUTPUT and FORWARD) have policies
- The policy can be either ACCEPT or DROP, for example:
- using the '-P' command
- e.g. # `iptables -P FORWARD DROP`

Managing rules

- Each rule specifies a set of conditions the packet must meet (matching condition),
and what to do if it meets them (‘target’ or action)
- For example
 - **to drop all ICMP packets coming from the IP address 127.0.0.1**
 - the conditions are that the protocol must be ICMP and that the source address must be 127.0.0.1
 - the target is ‘DROP’

➤ **to add the rule:** matching condition target

```
iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
```

➤ **to test:**
PING 127.0.0.1

➤ **to delete the rule:**
iptables -D INPUT 1 ,or
iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP



Filtering specifications

- Specifying an Interface
 - **the ` -i' (or `--in-interface') and ` -o' (or `--out-interface') options specify the name of an interface to match**
 - **node that:**
 - INPUT chain don't have an output interface
 - OUTPUT chain don't have an input interface
 - Only FORWARD chain have both an input and output interface
 - an interface name ending with a `+' (wildcard) will match all interfaces which begin with that string
- Specifying Source and Destination IP Addresses
 - **source (` -s', `--source' or `--src') and destination (` -d', `--destination' or `--dst') IP addresses can be specified in four ways**
 - using the full name, such as `localhost' or `www.linuxhq.com'
 - specifying the IP address, such as `127.0.0.1'
 - specifying a group of IP addresses, such as `199.95.207.0/24'
 - or such as `199.95.207.0/255.255.255.0'



Filtering specifications (cont.)

- Specifying Protocol
 - protocol can be specified with the '-p' (or '--protocol') flag
 - protocol can be a number or a name ('tcp', 'udp' or 'icmp')
- Specifying Inversion
 - many flags can have their arguments preceded by '!' (NOT) to invert (negate) the given matching condition
 - e.g. '-s ! localhost' matches any packet not coming from localhost



Matching extensions

- TCP, UDP and ICMP protocols automatically offer specific matching tests
 - **it is possible to specify the new match test on the command line after the '-p' option**
- E.g.
 - source-port (--sport) and --destination-port (--dport)**
 - followed by either a single TCP/UDP port, or a range of ports
 - ranges are two port names separated by a `:'
- Other extension can be loaded explicitly
 - **using the '-m' option followed by the match test**
 - **e.g. -m mac --mac-source 45:e4:23:6b:82:a0**



The state match

- The '-m state' extension interprets the connection-tracking analysis
 - **connection states are:**
 - NEW
 - a packet which creates a new connection
 - ESTABLISHED
 - a packet which belongs to an existing connection
 - RELATED
 - a packet which is related to, but not part of, an existing connection (e.g. an ICMP error, or an ftp data connection)
 - INVALID
 - a packet which could not be identified for some reason
- Example of '-m state' match extension:
 - `iptables -A FORWARD -i ppp0 -m state --state NEW -j DROP`

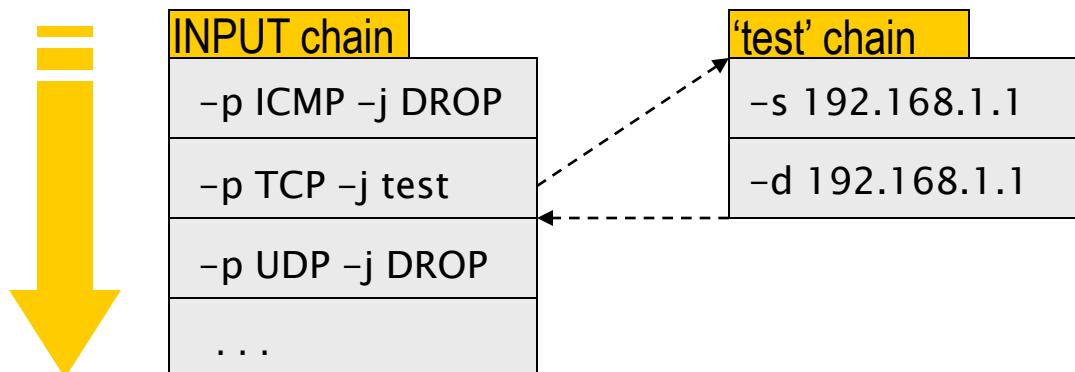


Target specifications

- Rule's target is what to do to the packets which match the rule
- There are two very simple built-in targets: DROP and ACCEPT
- Other targets are:
 - **LOG**
 - this module provides kernel logging of matching packets
 - **REJECT**
 - has the same effect as `DROP', except that the sender is sent an ICMP `port unreachable' error message
 - **RETURN**
 - has the same effect of falling off the end of a chain
 - **QUEUE**
 - is a special target, which queues the packet for userspace processing
 - **User-defined chains**

User-defined chains

- It is possible to create new chains, in addition to the three built-in ones (INPUT, FORWARD and OUTPUT)
- When a packet matches a rule whose target is a user-defined chain
 - the packet begins traversing the rules in that user-defined chain
 - if that chain doesn't decide the fate of the packet, then traversal resumes on the next rule in the current chain



Application-Layer Firewalls



Application-layer firewalls

- An application firewall is a form of firewall which controls input, output, and/or access from, to, or by an application or service
 - **can control all network traffic data up to the application layer**
 - it may inspect the contents of traffic, blocking specified content
 - such as certain websites, malicious programs, or attempts to exploit known logical flaws in client software
 - can restrict or prevent the spread of computer malwares
 - **unlike a stateful packet filter firewall which is (without additional software) unable to control network traffic regarding a specific application**
- There are two primary categories of application firewalls
 - **host-based application firewalls**
 - **network-based application firewalls**

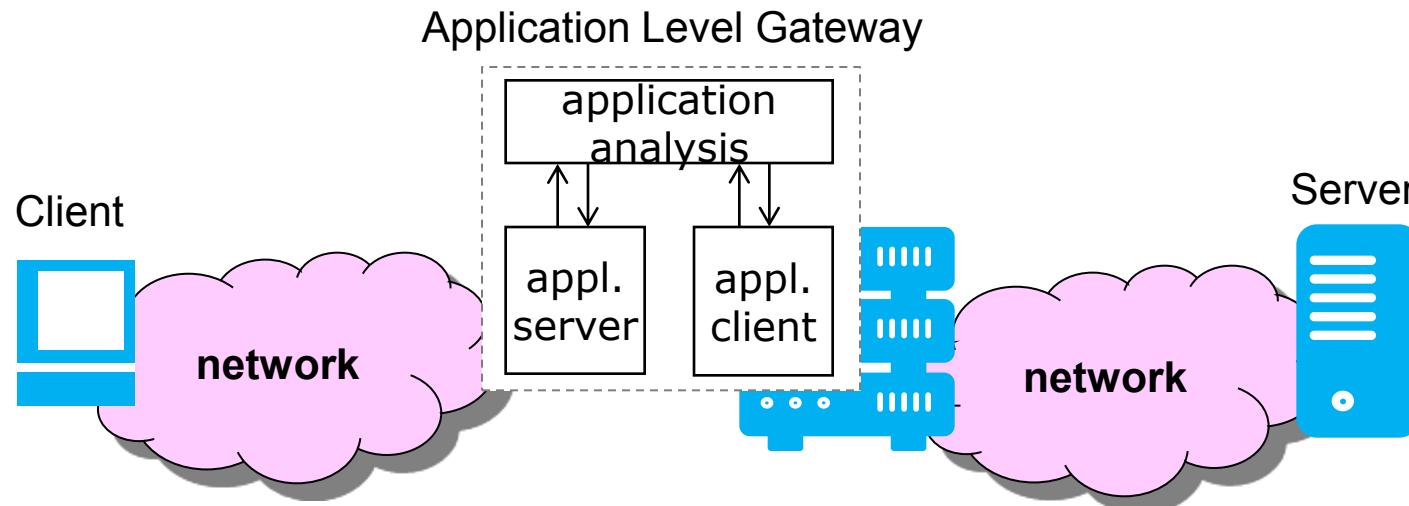


Host-based application firewalls

- A host-based application firewall provides protection to the applications running on the same host
- It can monitor any application input, output, and/or system calls made from, to, or by the application
 - **this is done by examining information passed through system calls instead of or in addition to a network stack**
 - **it may block the input, output, or system calls (including socket calls) which do not meet the configured policy of the firewall**
 - **they are able to apply filtering rules (allow/block) on a per process basis instead of filtering connections on a per port basis**
 - **generally, prompts are used to define rules for processes that have not yet received a connection**

Network-based application firewalls

- A network-based application layer firewall operates at the application layer of an (application) intermediate node
 - also known as proxy-based firewall or application-level gateway
 - it acts as a proxy/gateway for specific applications
 - specific to a particular kind of network traffic
 - e.g. HTTP and FTP proxy, SMTP server, SIP proxy, etc.





Network-based application firewalls (cont.)

- May run either as a stand-alone piece of network (dedicated) hardware, or as software on a general-purpose machine
 - often, it is a host using various forms of proxy servers to proxy traffic before passing it on to the client or server
- May run as single-homed host or on a dual-homed host



Network-based application firewalls (cont.)

- Advantages:

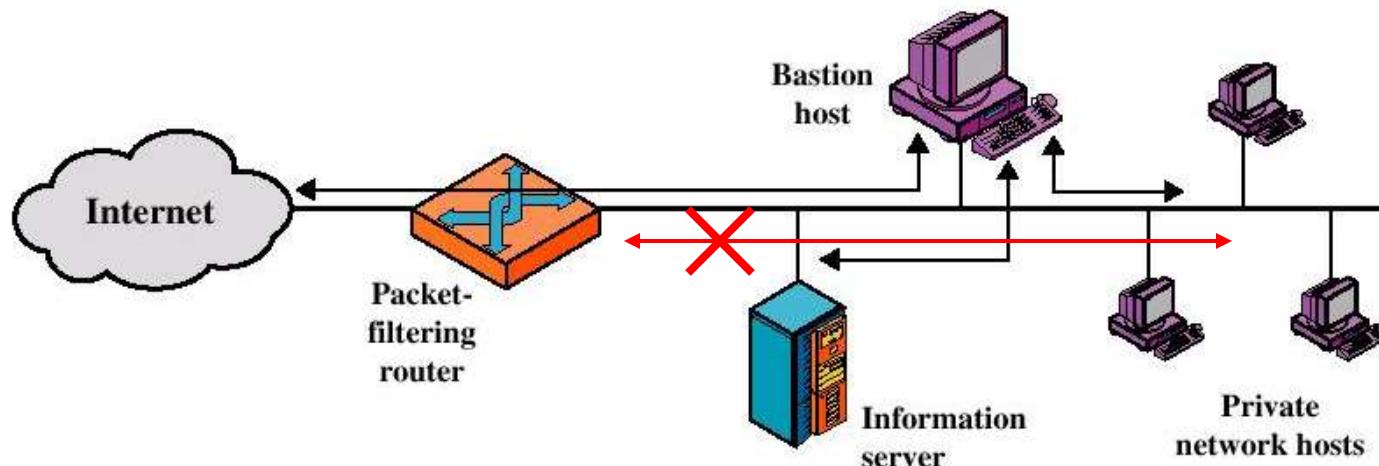
- **full control at application level**
 - content filtering
 - strong user authentication
 - higher level of security respect to a packet filter
- **easy to log and audit all incoming traffic**
- **by default, internal addresses are hidden**
- **caching**

- Disadvantages:

- **worse performances**
 - additional processing overhead on each connection
- **often requires explicit client configurations**
- **not suitable for new services/applications**
 - does support ONLY services for which a corresponding proxy is available (FTP, Telnet, HTTP, SMTP, ...)

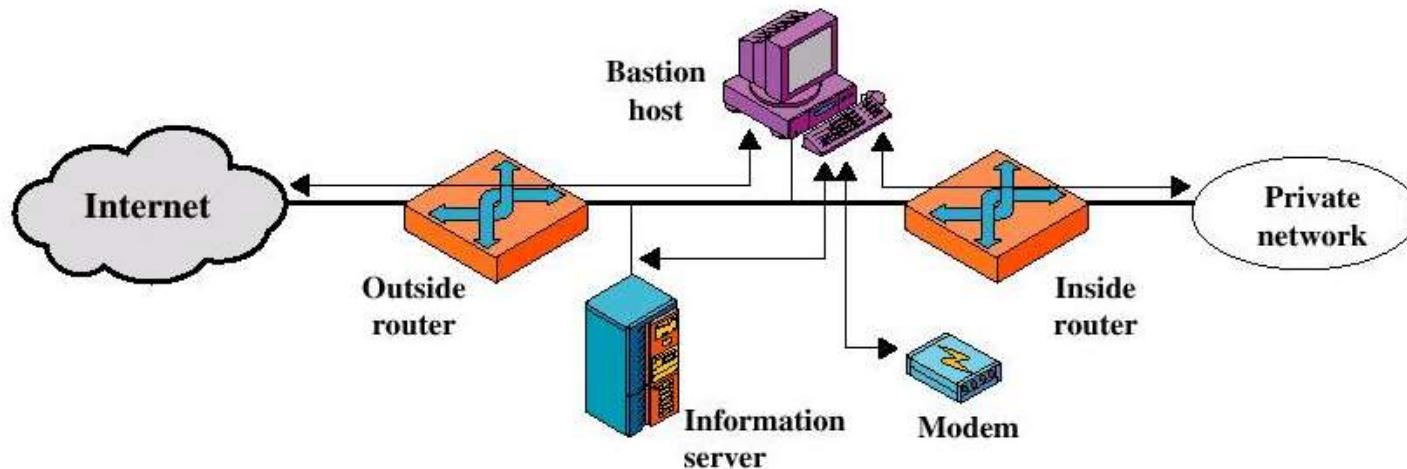
Hybrid configurations

Screened host firewall (single-homed bastion host)



- Screened host firewall, single-homed bastion configuration
- The firewall consists by two systems:
 - A **packet-filtering router**
 - A **bastion host**

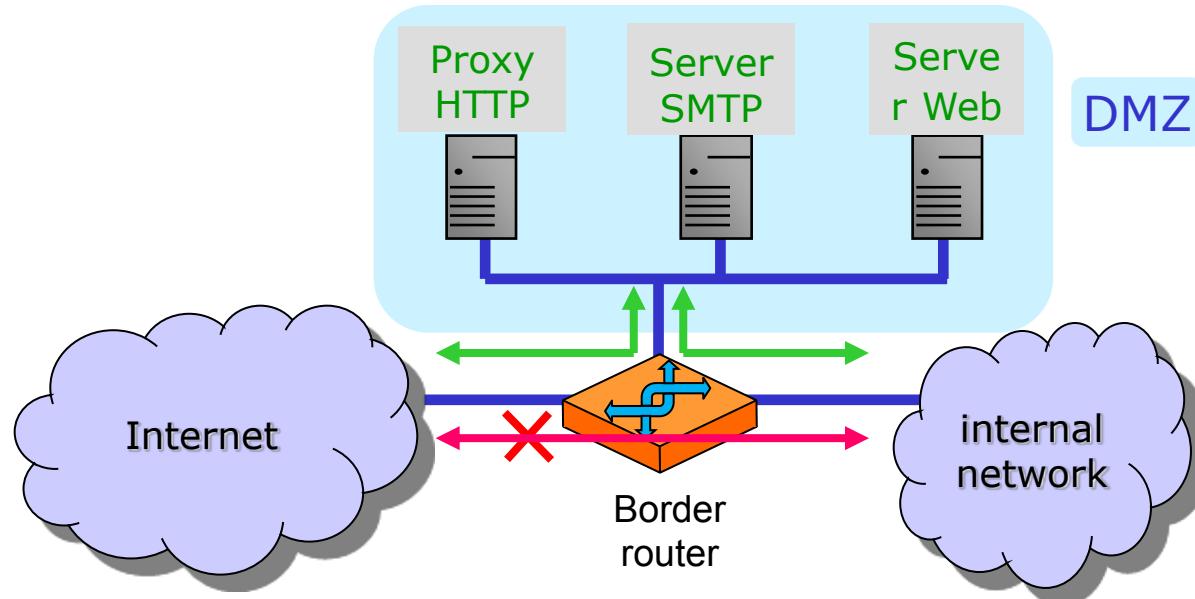
Screened-subnet firewall



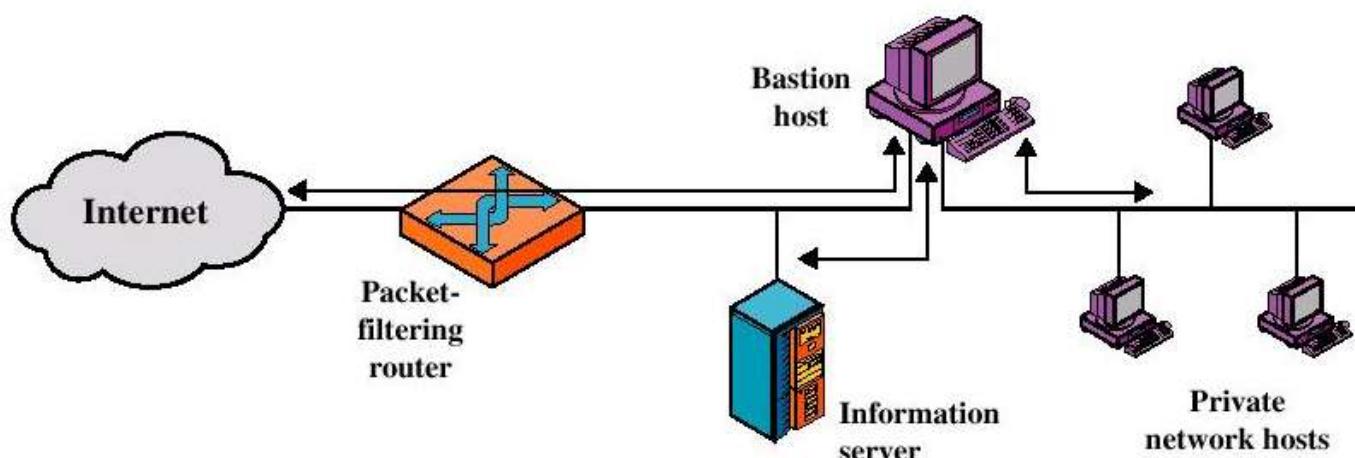
- Screened subnet firewall configuration
 - Most secure configuration
 - Two packet-filtering routers are used
 - Creation of an isolated sub-network (DMZ)

Screened-subnet firewall (cont.)

- Merging of interior/exterior router



Screened host firewall (dual-homed bastion host)



- Screened host firewall, dual-homed bastion configuration
 - **Traffic between the Internet and other hosts on the private network has to flow through the bastion host**
 - regardless the router configuration
 - **If the packet-filtering router is compromised, the network is still not completely compromised**



UNIVERSITÀ DI PARMA
Dipartimento di Ingegneria e Architettura

Intrusion Detection Systems

Luca Veltri

(mail.to: luca.veltri@unipr.it)

Course of Cybersecurity, 2022/2023

<http://netsec.unipr.it/veltri>



Intrusion Detection System (IDS)

- IDSs are software or hardware systems that automate the process of monitoring the events occurring in a computer system or network, analyzing them for signs of intrusions
 - **try to discover attempts to compromise or to bypass the security mechanisms of a computer or network**
 - **generate data as a consequence of normal or abnormal usage**
- IDSs process a stream of events E_1, E_2, E_3, \dots , and past system states S_1, S_2, S_3, \dots , and decide if a new event E_4 in S_4 is the final evidence that an intrusion is occurring
 - **they analyze the manifestation of an attack, not the result of the attack**
- An IDS may try to detect different types of intrusions:
 - **external attackers trying to access a system**
 - **authorized users of the systems who attempt to gain additional privileges for which they are not authorized**
 - **authorized users who misuse the privileges given them**



Network based IDS

- They detect attacks by capturing and analyzing network packets
 - monitoring a network segment or switch they can protect multiple host
- Often consist of a set of single-purpose nodes (called sensors) or hosts placed at various points in a network
 - sensor can run in “stealth” mode
- Majority of commercial IDSs
- Advantages:
 - few placed IDSs can monitor a large network
 - little impact upon an existing network
 - NIDSs are usually passive devices that listen on a network wire without interfering with the normal operation of a network
 - can be made very secure against attack and even made invisible to many attackers



Network based IDS

● Disadvantages

- **may have difficulty in processing all packets in a large or busy network**
 - HW implementation of a NIDS may help
- **switched networks**
 - networks are subdivided into many small segments (usually one wire per host)
 - most switches do not provide universal monitoring ports
- **cannot analyze encrypted information**
- **problems dealing with attacks that fragment packets**
- **often they cannot tell whether or not an attack was successful**
 - administrators must manually investigate each attacked host to determine whether it was indeed penetrated



Host based IDS

- Operate on information collected from within an individual computer system
 - application-based IDSs are actually a subset
 - great reliability and precision, determining exactly which processes and users are involved in a particular attack on the operating system
- Two types of information sources
 - operating system audit trails
 - usually generated at the innermost (kernel) level of the OS
 - more detailed and better protected than application logs
 - application logs
 - much smaller than OS trails
 - far easier to comprehend



Host based IDS

- Advantages
 - **detection of attacks that cannot be seen by a NIDS**
 - e.g. can help detect attacks involving software integrity holes
 - appear as inconsistencies in process execution
 - **they can “see” the outcome of an attempted attack**
 - they can directly access and monitor the data files and system processes usually targeted by attacks
 - **they are unaffected by switched networks and encrypted traffic**
- Disadvantages
 - **harder to manage, as information must be managed for every host monitored**
 - not well suited for detecting surveillance for an entire network
 - the amount of information can be immense
 - **use of the computing resources of the hosts they are monitoring**
 - **the IDS may be attacked and disabled as part of the attack (hosted by the systems it is monitoring)**



Challenges in Intrusion Detection

- Some challenges:
 - **Detect intrusion in real-time**
 - also in case of a huge stream of events
 - **Integrate different systems so that different analysis techniques and data source are covered**
 - e.g. data provided by network monitors and host auditing facilities
 - **Correlate detection results across different security domains**

Tools that complement IDS

- Vulnerability Analysis/Assessment Systems
 - tools to determine whether a network or host is vulnerable to known attacks
 - network-based (remote) analysis
 - testing by exploit
 - inference method (looking for the artifacts that successful attacks would leave behind)
 - e.g. Nessus, OpenVAS
- Honeypot System
 - system that look like a vulnerable system

